

## PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2001-236496

(43)Date of publication of application : 31.08.2001

(51)Int.Cl.

G06T 1/20  
G06F 15/16  
H03M 7/30

(21)Application number : 2000-342670

(71)Applicant : TEXAS INSTR INC &lt;TD&gt;

(22)Date of filing : 04.10.2000

(72)Inventor : CHIN-YUU HAN  
LEONARD W ESTEVEZ  
WISSAM A RABADIE

(30)Priority

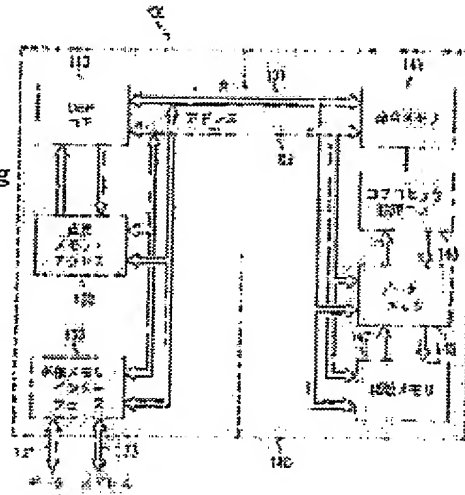
Priority number : 1999 411124 Priority date : 04.10.1999 Priority country : US

## (54) RECONSTRUCTIBLE SIMD CO-PROCESSOR STRUCTURE FOR SUMMING AND SYMMETRICAL FILTRATION OF ABSOLUTE DIFFERENCE

(57)Abstract:

PROBLEM TO BE SOLVED: To provide an image processing peripheral device which efficiently makes various calculations for image processing

SOLUTION: Proposed architecture is incorporated as a coprocessor 140 in a digital signal processor(DSP) and assists in the calculation of the total of absolute differences, symmetrical row/column FIR filtration having a down sampling (or up sampling) option, row/column discrete DCT/IDCT, and general algebraic functions. This architecture is composed of 8 multiplication accumulation hardware units, which are connected in parallel and have their paths selected and depends upon a DMA controller 120 to retrieve and write back data from and to a DSP memory without having a DSP core 110 intervene. The DSP after setting up DMA transfer and IPP/DMA synchronism in advance, moves to its process task. Furthermore, the DSP can be synchronized with IPP architecture to transfer and can synchronize data by itself.



(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2001-236496

(P2001-236496A)

(43) 公開日 平成13年8月31日 (2001.8.31)

(51) Int Cl.	識別記号	F I	特コード (参考)
G 0 6 I 1/20		G 0 6 T 1/20	B
G 0 6 F 15/16	6 1 0	G 0 6 F 15/16	6 1 0 A
H 0 3 M 7/30		H 0 3 M 7/30	Z

審査請求 未請求 請求項の数4 O L 外国語出願 (全 86 頁)

(21) 出願番号 特願2000-342670(P2000-342670)

(22) 出願日 平成12年10月4日 (2000.10.4)

(31) 優先権主張番号 4 1 1 1 2 4

(32) 優先日 平成11年10月4日 (1999.10.4)

(33) 優先権主張国 米国 (U S)

(71) 出願人 590000879

テキサス インスツルメンツ インコーポ  
レイテッド  
アメリカ合衆国テキサス州ダラス、ノース  
セントラルエクスプレスウェイ 13500

(72) 発明者 テン - ユー - ハン

アメリカ合衆国 テキサス、プラノ、ポー  
ルドウィンレーン 4633

(74) 代理人 100066692

弁理士 浅村 皓 (外3名)

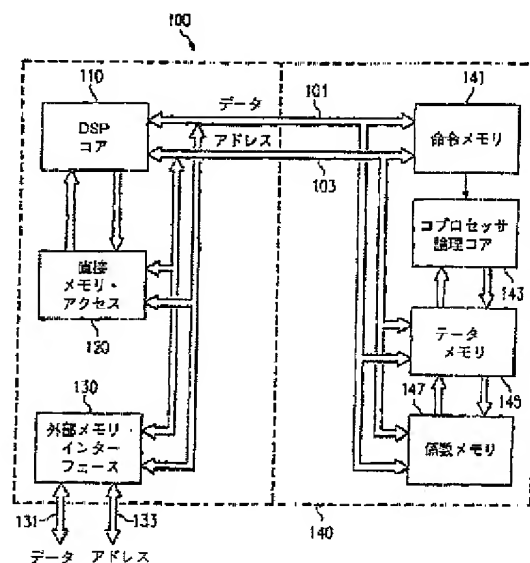
最終頁に続く

(54) 【発明の名称】 絶対差分の合計および対称濾波用の再構成可能S I M Dコプロセッサ構造

(57) 【要約】

【課題】 画像処理用の種々の計算を効率的に行う画像処理周辺装置を提供する。

【解決手段】 提案されるアーキテクチャは、コプロセッサ140としてデジタル信号プロセッサ (DSP) に組み込まれて、絶対差分の合計の計算、ダウンサンプリング (またはアップサンプリング) オプションを持つ対称行/列F I R濾波、行/列離散D C T / I D C Tおよび一般代数機能アシストする。このアーキテクチャは、並列に接続されて経路選択され多重化された8個の乗算累算ハードウェア・ユニットからなり、DMAコントローラ120に依存し、DSPコア110が介入することなくDSPメモリとの間でデータの検索および書戻しを行う。DSPは、予めDMA転送およびI P P / D M A同期をセットアップしたのち、それ自身の処理タスクに移る。または、DSPは、I P Pアーキテクチャと同期をとって自身でデータの転送および同期を行うことができる。



## 【特許請求の範囲】

【請求項 1】 画像処理周辺装置であって、

並列に接続された乗算累算回路の複数の対を具備し、  
該乗算累算回路の各対が、

複数の第 1 の加算器対であって、各加算器対の各加算器が、第 1 の所定数のビットを有する第 1 および第 2 の入力をそれぞれ受ける第 1 および第 2 の入力と、前記入力の和または差を生成する出力とを有する、複数の第 1 の加算器対と、

該複数の第 1 の加算器対に対応する複数の第 1 の乗算器対であって、各乗算器対の各乗算器が、前記第 1 の加算器の前記和または差の第 1 の入力と、一定の所定数の第 2 の入力を有するとともに、積出力を生成する、複数の第 1 の乗算器対と、

該複数の第 1 の乗算器対に対応する複数の第 2 の加算器対であって、該加算器対の各加算器が、前記対応する乗算器対の前記乗算器の一方または他方から第 1 の乗算器出力を前記第 1 の入力としてそれぞれ受ける第 1 および第 2 の入力を有する、第 2 の加算器対と、

を備え、  
前記第 2 の加算器の対の前記一方がマルチプレクサからの出力を受け、該マルチプレクサが、前記第 1 の乗算器対の前記他方の乗算器の積からの第 1 の入力を有するとともに、前記第 2 の加算器対の前記一方の加算器の累算された和からの第 2 の入力を前記第 2 の加算器対の前記一方の加算器の第 2 の入力として有し、

前記第 2 の加算器の対の前記他方が第 1 および第 2 のマルチプレクサからの出力を受け、前記第 1 のマルチプレクサが前記第 1 の乗算器対の前記他方の乗算器からの第 1 の入力と前記第 2 の加算器対の前記一方の加算器の和からの第 2 の入力を有し、前記第 2 のマルチプレクサが前記第 2 の加算器対の前記他方の加算器の累算された和からの第 1 の入力を有するとともに第 2 の加算器対の第 2 の対の一方の加算器の和からの第 2 の入力を第 2 の出力として有し、

前記第 2 の加算器対が和出力を生成する

画像処理周辺装置。

【請求項 2】 画像処理周辺装置であって、

並列に接続された乗算累算回路の複数の対を具備し、

該乗算累算回路の各対が、

複数の第 1 の加算器対であって、各加算器対の各加算器が、第 1 の所定数のビットを有する第 1 および第 2 の入力をそれぞれ受ける第 1 および第 2 の入力と、該入力の和または差を生成する出力とを有する、複数の第 1 の加算器対と、

該複数の第 1 の加算器対に対応する複数の第 1 の乗算器対であって、各乗算器対の各乗算器が、前記第 1 の加算器の前記和または差の第 1 の入力と一定の所定数の第 2 の入力を有するとともに、積出力を生成する、複数の第 1 の乗算器対と

該複数の第 1 の乗算器対に対応する複数の第 2 の加算器対であって、各加算器対が、前記第 1 の乗算器対の前記積の別々の累算を実行し、累算された合計を生成する、複数の第 2 の加算器対と、

を備える、

画像処理周辺装置。

【請求項 3】 画像処理周辺装置であって

並列に接続された乗算累算回路の複数の対を具備し、

該乗算累算回路の各対が、

複数の第 1 の加算器対であって、各加算器対の各加算器が、第 1 の所定数のビットを有する第 1 および第 2 の入力をそれぞれ受ける第 1 および第 2 の入力と、該入力の和または差を生成する出力とを有する、複数の第 1 の加算器対と、

該複数の第 1 の加算器対に対応する複数の第 1 の乗算器対であって、各乗算器対の各乗算器が、前記第 1 の加算器の前記和または差の第 1 の入力と一定の所定数の第 2 の入力を有するとともに、積出力を生成する、複数の第 1 の乗算器対と、

該複数の第 1 の乗算器対に対応する複数の第 2 の加算器対であって、各加算器の対が、前記乗算器の対の前記積の加算を実行したのち、該加算の和を累算する、複数の第 2 の加算器対と、

を備える、

画像処理周辺装置。

【請求項 4】 画像処理周辺装置であって、

並列に接続された乗算累算回路の複数の対を具備し、

該乗算・累算回路の各対が

複数の第 1 の加算器対であって、各加算器対の各加算器が、第 1 の所定数のビットを有する第 1 および第 2 の入力をそれぞれ受ける第 1 および第 2 の入力と、該入力の和または差を生成する出力とを有する、複数の第 1 の加算器対と

該複数の第 1 の加算器対に対応する複数の第 1 の乗算器対であって、各乗算器対の各乗算器が、前記第 1 の加算器の前記和または差の第 1 の入力と一定の所定数の第 2 の入力を有するとともに、積出力を生成する、複数の第 1 の乗算器対と、

該複数の第 1 の乗算器対に対応する複数の第 2 の加算器対であって、各加算器対が 2 つの積の和の累算を実行する、複数の第 2 の加算器対と、

を備える

画像処理周辺装置。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、一般に、信号処理に関し、より詳しくは、一次元および二次元濾波、変換および他の共通タスクを含む高速画像およびビデオ信号処理を行う単一命令多重データ (SIMD) ・コプロセッサ・アーキテクチャに関する。

## 【0002】

【従来の技術】画像処理技術でこれまで起こっている問題は、二次元（2-D）濾波が一次元（1-D）濾波とは異なるアドレス指定パターンを有することである。1-D用に設計された従来のDSPプロセッサおよびコプロセッサは、2-Dビデオ信号を処理するためには変更されなければならない。最終目的は、デジタル信号プロセッサ（DSP）またはコプロセッサが画像およびビデオ処理を適切に行えるようにすることである。画像処理では、最も有用な操作は1-Dおよび2-D濾波であって、これは2-Dデータと1-Dまたは2-D畳込み係数とをアドレス指定する必要がある。畳込み係数が対称であるときには、対称性を利用するアーキテクチャは計算時間をほぼ半分に減らすことができる。多くのビデオ符号化アルゴリズムに存在する主なボトルネックは、動き推定のそれである。動き推定の問題に対処するには、まず画像と核（kernel）とをコンボルブして（convolve）、それを低解像度の画像にする。次に、これらの画像が同じ核で再びコンボルブされ、さらに低解像度の画像を生成する。次に、絶対差分の合計が各レベルにおいてサーチ窓内で計算されて、前フレームの副画像と最もよく一致する副画像を決定する。より低い解像度で最もよい一致が得られると、サーチがより高い解像度で対応近傍内で繰り返される。上記から、1-D/2-D濾波（好ましくは、対称濾波も）と絶対差分の合計とを同じ効率で行うことができるアーキテクチャを作る必要が生じた。従来は、SIMDコプロセッサ・アーキテクチャで絶対差分の合計および対称濾波の動作を行うのに、特殊ハードウェアまたは汎用DSPが用いられている。インテルのMMX技術は、汎用性は高いが、概念は似ている。1998年2月4日時点で同時係属中の出願である「多重乗算累算（多重MAC）コプロセッサ・アーキテクチャ」シリアル番号第60/073 668号（TI-26868）と「効率的に接続されたハードウェアコプロセッサを持つDSP」シリアル番号第60/073 641号（TI-26867）とは、ホスト・プロセッサ/コプロセッサ・インターフェースと効率的な有限インパルス応答/高速フーリエ変換（FIR/FFT）濾波とを実現しており、本発明はこれを拡張していくつかの他の機能を付加する。

## 【0003】

【課題を解決するための手段】ここに提案するアーキテクチャは、コプロセッサとしてデジタル信号プロセッサ（DSP）に組み込まれて、絶対差分の合計、ダウンサンプリング（またはアップサンプリング）、オプションを持つ対称な行/列有限インパルス応答（FIR）濾波、行/列離散コサイン変換（DCT）/逆離散コサイン変換（IDCT）および一般代数関数の計算を支援する。このアーキテクチャは、画像処理周辺装置を表すIPPと呼ばれ、並列に接続された、経路選択された

多重化された8個の乗算累算ハードウェア・ユニットからなる。このアーキテクチャは、直接メモリ・アクセス（DMA）・コントローラに依存し、DSPコアが介入することなくDSPメモリとの間でデータの検索および書込みを行う。DSPは、予めDMA転送およびIPP/DMA同期をセットアップし、それ自身の処理タスクに移る。または、DSPは、これらの転送についてIPPアーキテクチャと同期することによって、自身でデータの転送および同期を行うことができる。このアーキテクチャは、既に開示されている多重MACコプロセッサ・アーキテクチャ（TI-26868 シリアル番号第60/073 641号、「変更可能多重乗算累算ハードウェア・コプロセッサ・ユニット」、1998年1月4日出願、ここに援用する）よりも、2-D濾波、対称濾波、短フィルタ、絶対差分の合計およびモザイク復号を効率的に行うことができる。このコプロセッサは、特に共通2-D信号処理タスクを行うDSPの機能を大幅に増進する。このアーキテクチャは拡張可能であって、

（DMAがDSPとコプロセッサとの間のデータ転送を十分高速で処理することができれば）このアーキテクチャに単一命令多重データ（SIMD）ブロックを追加する度に性能を高めることができる。この技術はビデオ符号化を大幅に増進する。このアーキテクチャは、テキサスインスツルメンツ社のTMS320C54xやTMS320C6xのような既存のDSPに組み込むことができる。これらのプロセッサはそれぞれ、データ転送用のDMAコントローラを既に内蔵している。

【0004】本明細書の一部を構成するものとして挿入する添付の図面は、本発明の好ましい実施の形態の略図であり、好ましい実施の形態の一般的な説明および詳細な説明と共に本発明の原理を明らかにするものである。

## 【0005】

【発明の実施の形態】図1は、デジタル信号プロセッサ・コア110と再構成可能IPPハードウェア・コプロセッサ140とを含む回路100を示す。図1は、同じ被譲渡人に譲渡された同時係属中の出願番号第60/073 641号「変更可能多重乗算累算ハードウェア・コプロセッサ・ユニット」の図1と同じであって、本発明の好ましい実施の形態はこのコプロセッサを用いる。本発明の好ましい実施の形態によれば、これらの構成部品は単一集積回路（IC）に形成される。デジタル信号プロセッサ・コア110は従来の設計のものでよい。IPPはメモリ・マップ周辺装置である。IPPの作業メモリとDSPの作業メモリとの間のデータの転送は、デジタル信号プロセッサ・コア110が介入することなく、直接メモリ・アクセス（DMA）・コントローラ120を介して行うことができる。または、DSPコア110は、IPPの作業メモリ141、145、147に直接ロード/記憶することにより自身でデータ転送を処理することができる。DMAは大量のデータ/係



数転送をより効率的に処理することができる。また、DSPは短い命令をIPP命令メモリ141に一層効率的に直接書き込むことができるので、この2つの転送機構を結合することも可能である。

【0006】再構成可能IPPハードウェア・コプロセッサ140は、広範囲の機能性を有し、対称／非対称・行／列濾波・2-D濾波・絶対差分の合計・行／列DCT／IDCTおよび一般線形代数関数を支援する。対称行／列濾波は、アップ／ダウン・サンプリングによく用いられ、ディスプレイ装置に適合するように画像の大きさを調整する。2次元濾波は、デジタル・カメラにおけるデモザイクおよび画像強調にしばしば用いられる。絶対差分の合計は、MPEGビデオ符号化とH.263およびH.323（電話線ビデオ会議用の符号化標準）とに用いられる。行／列DCT／IDCTは、JPEG画像符号／復号およびMPEGビデオ符号／復号に用いられる。配列加算／減算および基準化を含む一般線形代数関数は、画像およびビデオ応用によく用いられ、濾波および変換動作を補足する。例えば、デジタル・カメラは、利得制御とホワイト・バランスとを行うのに画素を基準化する必要がある。

【0007】好ましい実施の形態では、再構成可能IPPハードウェア・コプロセッサ140は、デジタル信号プロセッサ・コア110と無関係な自動データ転送のために直接メモリ・アクセス回路120と協働するようにプログラムされることができる。外部メモリ・インターフェース130は、内部データ・バス101およびアドレス・バス103をそれらの外部の対応する外部データ・バス131および外部アドレス・バス133にそれぞれインターフェースする。外部メモリ・インターフェース130は従来の構造のものである。集積回路100は、別の従来の機能および回路を任意に含んでもよい。キャッシュ・メモリを集積回路100に追加すれば性能が大きく改善されることに特に留意すべきである。図1に示す構成要素は従来の構成要素の提供を除く意図ではない。図1に示す従来の構成要素は、変更可能ハードウェア・コプロセッサ140の追加によって最も影響を受ける構成要素にすぎない。

【0008】再構成可能IPPハードウェア・コプロセッサ140は、データ・バス101およびアドレス・バス103を介して集積回路100の他の構成要素に結合されている。再構成可能IPPハードウェア・コプロセッサ140は、命令メモリ141とコプロセッサ論理コア143とデータ・メモリ145と係数メモリ147とを含む。命令メモリ141は、デジタル信号プロセッサ・コア110が再構成可能IPPハードウェア・コプロセッサ140の動作を制御するときに中継の働きをする。コプロセッサ論理コア143は、命令待ち行列を形成してコプロセッシング機能 (co-processing functions) を行う命令メモリ141に記憶された命令に応答す

る。これらのコプロセッシング機能は、コプロセッサ論理コア143とデータ・メモリ145と係数メモリ147との間のデータの交換を含む。データ・メモリ145は、変更可能ハードウェア・コプロセッサ140によって処理された入力データを記憶するとともに、変更可能ハードウェア・コプロセッサ140の動作の結果を記憶する。係数メモリ147は、コプロセッサ論理コア143によって用いられる係数と呼ばれる不変のまたは比較的不变のプロセス・パラメータを記憶する。データ・メモリ145と係数メモリ147とは別個の構成要素として示されているが、これらを単に一個の統一メモリの異なる部分として用いることもできる。後で示すように、上述した多重乗算累算コプロセッサでは、かかる1個の統一メモリがデータおよび係数を読み取る2個の読取りポートと出力データを書き込む1個の書込みポートとを有するならば、それがベストである。多重ポート・メモリは同じ容量の単一ポート・メモリよりも大きなシリコン領域を必要とするので、メモリ・システムは複数のブロックに分割されて、多重アクセス・ポイントを達成することができる。かかるメモリ構成では、メモリ・アクセス衝突を処理するために、メモリ調停および停止機構を備えたIPPを装備することが望ましい。再構成可能IPPハードウェア・コプロセッサ140によってアクセスすることのできるメモリは同じ集積回路上にコプロセッサ論理コア143に物理的に近接して置くのがベストであると考えられる。コプロセッサ論理コア143の所望のデータ・スループットによって要求される広いメモリ・バスを設けるためには、この物理的な近接が必要である。

【0009】図2は、デジタル信号プロセッサ・コア110と再構成可能IPPハードウェア・コプロセッサ140との間のメモリ・マップ・インターフェースを示す。デジタル信号プロセッサ・コア110は命令メモリ141を介して再構成可能IPPハードウェア・コプロセッサ140を制御する。好ましい実施の形態では、命令メモリ141は、命令待ち行列を持つ先入れ先出し (FIFO) メモリである。命令メモリ141の書込みポートは、デジタル信号プロセッサ・コア110のアドレス空間内の単一メモリ位置にメモリ・マッピングされる。このように、デジタル信号プロセッサ・コア110は、命令メモリ141への入力となるアドレスに命令を書き込むことによって再構成可能IPPハードウェア・コプロセッサ140を制御する。命令メモリ141は、好ましくは、2個の循環的に方向付けられたポイントを含む。書込みポイント151は、次に受ける命令が記憶されるべき命令メモリ141内の位置を指し示す。命令メモリ141の所定のアドレスに書込みがある度に、書込みポイントは、データを受ける物理的位置を選択する。かかるデータ書込みが終わると、書込みポイント151は更新されて命令メモリ141内の次の物理的

位置を指し示す。書き込みポインタ151は、最後の物理的位置から最初の物理的位置に循環する点で、循環的に方向付けられている。再構成可能IPPハードウェア・コプロセッサ140は、命令を、それらが読取りポインタ153を用いて受け取られた(FIFO)のと同じ順序で命令メモリ141から読み取る。読取りポインタ153は、読み取られるべき次の命令を記憶する命令メモリ141内の物理的位置を指し示す。読取りポインタ153は更新されてそのような読取りに続く命令メモリ141内の次の物理的位置を指し示す。読取りポインタ153も循環的に方向付けられており最後の物理的位置から最初の物理的位置に循環することに留意すべきである。命令メモリ141は、書き込みポインタ151が読取りポインタ153を通るのを妨げる機能を含む。この機能により、例えば、書き込みポインタ151と読取りポインタ153とが同じ物理的位置を指し示したとき、書き込みを拒否して、メモリ故障信号をデジタル信号プロセッサ・コア110に送り返す。命令メモリ141のFIFOバッファは一杯になってそれ以上の命令を受け付けなくなることがある。

【0010】多くのデジタル信号処理タスクは同様な機能の複数の事例を用いる。例えば、処理は複数のフィルタ機能を含むことがある。再構成可能IPPハードウェア・コプロセッサ140はこれらのフィルタ機能の全てを実時間で行うだけの十分な処理容量を持つのが好ましい。マクロ記憶領域149はサブルーチンの形で共通機能を記憶するのに用いられ、これらの機能を行うには命令待ち行列141の「サブルーチン呼出し」命令を用いるだけでよい。これは、命令メモリ上のトラヒックと命令メモリ上の潜在的なメモリ必要量を全体的に減少させる。図2は、各サブルーチンが「リターン」命令で終わる3つのサブルーチンA、B、Cがマクロ記憶領域149上に常駐していることを示す。

【0011】命令FIFO/マクロ記憶組合せに代わるものは、DSPが最初にセットアップする静的命令メモリ内容である。命令メモリは、それぞれが「スリープ」命令で終わる多数の命令シーケンスを保持することができる。DSPは、シーケンスの開始アドレスをIPP制御レジスタに書き込むことによって特定の命令シーケンスを実行するようにIPPに命じる。IPPは、DSPから別の命令が来るのを待つ待機モードに入ると、スリープ命令が来るまで、指定された命令を実行する。データ・メモリ145と係数メモリ147とは共にデジタル信号プロセッサ・コア110のデータ・アドレス空間内にマッピングされる。図2に示すように、データ・バス101はメモリ149に双方向に結合される。上述した代替の実施の形態では、データ・メモリ145と係数メモリ147とは共にメモリ149の一部として形成される。メモリ149はまた、(図2には示していない)コプロセッサ論理コア143によってアクセス可能であ

る。図2は、メモリ149内のメモリの3つの線で囲まれた領域を示す。後で更に説明するが、再構成可能IPPハードウェア・コプロセッサ140は、異なるメモリ領域を用いていくつかの機能を実行する。

【0012】集積回路100は次のように動作する。デジタル信号プロセッサ・コア110またはDMAコントローラ120は、データをデータ・メモリ145に係数を係数メモリ147に、または、データおよび係数を統一メモリ149にロードすることにより、再構成可能IPPハードウェア・コプロセッサ140によって用いられるデータおよび係数を制御する。デジタル信号プロセッサ・コア110はこのデータ転送を直接行うようにプログラムされてもよいし、デジタル信号プロセッサ・コア110はDMAコントローラ120を制御してこのデータ転送を行うようにプログラムされてもよい。特にオーディオまたはビデオ処理の応用では、データ・ストリームは予測可能な速度で予測可能な装置から受信される。このように、デジタル信号プロセッサ・コア110がDMAコントローラ120を制御して外部メモリから再構成可能IPPハードウェア・コプロセッサ140によってアクセス可能なメモリに転送させることが、一般に、効率的である。

【0013】処理されるべきデータの転送後、デジタル信号プロセッサ・コア110は所望の信号処理アルゴリズム用の命令を用いて再構成可能IPPハードウェア・コプロセッサ140に知らせる。前述したように、命令は、命令待ち行列141内の所定のアドレスへのメモリ書き込みによって再構成可能IPPハードウェア・コプロセッサ140に送られる。受け取られた命令は命令待ち行列141に先入れ先出し方式で記憶される。再構成可能IPPハードウェア・コプロセッサの各計算命令は、好ましくは、実行されるべき特定の機能を指定する方法を含む。好ましい実施の形態では、再構成可能IPPハードウェア・コプロセッサは再構成可能なように構築されている。再構成可能IPPハードウェア・コプロセッサは、乗算器や加算器のような、異なるが関連する機能を行うように種々の方法で互いに接続することができる一組の機能ユニットを有する。再構成可能IPPハードウェア・コプロセッサ毎に選択される関連する機能の集合は、機能の数学的類似性に基づく。この数学的類似性により、同様なハードウェアを複数の機能に用いるように再構成することができる。命令は、データ・プロセッサ命令の方法でオペコードを介して特定の計算を指示してもよい。

【0014】各計算命令は、計算に用いられる入力データの位置を指定する方法を含む。データ空間を指定する適当な方法は多数ある。例えば、命令はそのブロック内のデータ語またはサンプルの開始アドレスおよび数を指定する。データサイズは、パラメータとして指定されてもよく、計算タイプを定義するオペコードで指定されて

もよい。別の例を挙げると、命令は入力データのデータサイズ、開始アドレスおよび終了アドレスを指定する。入力データが記憶されている位置を指定する既知の間接法を用いてもよい。命令は、データ・ブロック内のサンプルの開始アドレス、データサイズおよび数と終了アドレスとのような任意の数のこれらのパラメータを記憶するレジスタまたはメモリ位置を指し示すポインタを含んでもよい。

【0015】各計算命令は、特定の命令の出力データを記憶するメモリ・アドレス範囲をさらに指示しなければならない。この指示は、入力データを記憶する位置に関して前に列挙されたの方法を用いて行ってもよい。多くの場合、計算機能は簡単なフィルタ機能であり、処理後の出力データの量は入力データの量にほぼ等しい。他の場合には、出力データの量が入力データの量よりも多かったり少なかったりする。どちらにしても、得られるデータの量は、入力データの量と要求される計算機能の型とから分かる。したがって、単に開始アドレスを指定するだけで、得られる全てのデータを記憶する場所を示す十分な情報が与えられる。処理中に入力データに上書きする破壊的な方法で出力データを記憶してもよい。または、出力データをメモリの別の場所に書き込んで、入力データを少なくとも一時的に保存してもよい。これらの種々の方法のどれを選択するかは、入力データを再使用するかどうかに依存する。

【0016】図3は、2つのメモリ領域を交互に用いることを含む有用な方法を示す。一方のメモリ領域145はコプロセッサ機能に必要な入力データを記憶する。比

```
Row_filter(us, ds, length, block, data_addr, coef_addr, outp_addr)
Column_filter(us, ds, length, block, data_addr, coef_addr, outp_addr)
Row_filter_sym(us, ds, length, block, data_addr, coef_addr, outp_addr)
Sum_abs_diff(length, data_addr1, data_addr2, outp_addr)
Row_DCI(data_addr, outp_addr), Row_IDCI, Column_DCI, Column_IDCI
Vector_add(length, data_addr1, data_addr2, outp_addr)
```

【0019】これらのパラメータは、所望の量を記憶するレジスタまたはメモリ位置へのポインタである直接量または変数の形をとる。これらのパラメータの数およびタイプは命令タイプに依存する。このサブルーチン呼出し書式は、デジタル信号プロセッサ・コア110用に書かれたプログラムを再使用するのに重要である。用いるときは、プログラマまたはコンパイラは、スタブ・サブルーチンを与えて、再構成可能IPPハードウェア・コプロセッサ140を活動化する。このスタブ・サブルーチンは、単にサブルーチン・パラメータを受け取りこれらのパラメータを用いて対応コプロセッサ命令を形成する。次に、スタブ・サブルーチンは、再構成可能IPPハードウェア・コプロセッサ140への命令転送用に予約された所定のメモリ・アドレスにこの命令を書き込んだのち、復帰する。本発明は、デジタル信号プロ

較的一定の係数は係数メモリ147に記憶される。入力データは、コプロセッサ論理コア143によって用いられるためにデータ・メモリ145の第1のメモリ領域144から取り出される（1読取り）。出力データは、データ・メモリの第2のメモリ領域146に書き込まれる（1書込み）。データ・メモリ領域の使用後、直接メモリ・アクセス回路120は、データを第1のメモリ領域144の次のブロックに書き込み、前に用いられたデータを上書きする（2書込み）。同時に、直接メモリ・アクセス回路120は、データを、再構成可能IPPハードウェア・コプロセッサ140によって上書きされる前に第2のメモリ領域146から読み取る（2読取り）。入力データ用および得られたデータ用のこれらの2つのメモリ領域は循環バッファとして構成されてもよい。複数の関連機能が必要とする積では、循環バッファと定義された別々のメモリ領域が用いられてもよい。循環バッファとして構成された1つのメモリ領域は別々の機能に割り当てられる。

【0017】計算命令の書式は、好ましくは、高レベル言語のサブルーチン呼出し命令の書式にかなり似ている。すなわち、命令は、実行されるべき特定の計算機能を指定するサブルーチン名と機能的に似た命令名を含む。各命令はまた、命令タイプ内の利用可能オプションを指定する一組のパラメータを含む。例えば、計算命令およびさまざまなパラメータの以下のリスト。

【0018】

【表1】

セッサ・コアの計算容量が時間とともに規則的に増加することをもくろむ。このように、ある積の処理要求はある時点でのデジタル信号プロセッサ・コア110と再構成可能IPPハードウェア・コプロセッサ140との結合を必要とする。後の時点では、デジタル信号プロセッサ・コアの命令集合の利用可能計算容量が増加するので、以前に再構成可能IPPハードウェア・コプロセッサを必要とした機能はデジタル信号プロセッサ・コアによってソフトウェアで行われる。その積用の前のプログラム・コードは、新しい層強力なデジタル信号プロセッサに容易に変換され得る。これは、置換された再構成可能ハードウェア・コプロセッサによってサポートされる命令毎に独立サブルーチンを与えることによって達成される。次に、原プログラムがサブルーチン・スタブを用いて命令を送る各場所が、対応サブルーチン

呼出しによって置き換えられる。このようにして、大規模なプログラミングの書き換えを避けることができる。

【0020】1つのデータのブロックの処理が終わると、データはデータ・メモリ145から転送される。この第2の転送は、出力メモリ位置に記憶されているデータを読み取るデジタル信号プロセッサ・コア110の直接動作、または、直接メモリ・アクセス回路120の助けを借りて、行われる。この出力データは処理の出力を表してもよい。この場合は、データは利用装置に転送される。または、再構成可能I P Pハードウェア・コプロセッサ140の出力データは、進行中の作業を表してもよい。この場合には、データは、一般に、後の検索および更なる処理のために集積回路100の外部のメモリに一時的に記憶される。

【0021】その後、再構成可能I P Pハードウェア・コプロセッサ140は更なる使用の準備をする。この更なる使用は同じ機能の別の処理でもよい。この場合には、上述した過程がデータの新しいブロックについて同じ方法で繰り返される。この更なる使用は他の機能の処理でもよい。この場合には、データの新しいブロックが再構成可能I P Pハードウェア・コプロセッサ140によってアクセス可能なメモリにロードされ、新しい命令がロードされたのち、処理されたデータが出力または更なる処理のために読み取られなければならない。

【0022】再構成可能I P Pハードウェア・コプロセッサ140は、好ましくは、積アルゴリズムの2つ以上の機能を行うことができるであろう。離散サンプルではなくデータのブロックについて操作する利点は、再構成可能I P Pハードウェア・コプロセッサ140がかかるシステムで動作するときに明らかになるであろう。一例として、再構成可能I P Pハードウェア・コプロセッサ140が3つの機能A、B、Cを行うとする。これらの機能は、一連のものでもよいし、デジタル信号プロセッサ・コア110によって行われる機能とインターリーブされてもよい。再構成可能I P Pハードウェア・コプロセッサ140は、まず、データのブロックに対して機能Aを行う。この機能は上述したように行われる。デジタル信号プロセッサ・コア110は、直接、または、直接メモリ・アクセス回路120の制御によって、入力データをデータ・メモリ145にロードする。処理されるべきデータの量を指定する機能Aの構成用の命令が出されると、再構成可能I P Pハードウェア・コプロセッサ140は、機能Aを行い、命令によって指定されたメモリ145の部分に得られた結果を戻して記憶する。同様の処理が生じて、再構成可能I P Pハードウェア・コプロセッサ140に、メモリ145に記憶されているデータに対して機能Bを行わせ、その結果をメモリ145に戻させる。機能Aの実行は、機能B用のデータ・ブロックのサイズと関係がないサイズを有するデータ・ブロックについて起こる。最後に、再構成可能I P Pハード

ウェア・コプロセッサ140は、メモリ145内のデータに対して機能Cを行うように命令され、その結果をメモリ145に戻す。機能Cを行うブロックサイズは、機能A、B用に選択されたブロックサイズと関係がない。

【0023】この例から、ブロック処理の有用性が分かる。3つの機能A、B、Cは、一般に、機能間のフィルタ履歴およびアップ/ダウン・サンプリングにより実際の入力/出力サイズと必ずしも等しくない1つの共通のデータ処理サイズ（例えば、最終出力として1つの16×16画素ブロック）に関する作業量を行う。機能毎に特殊なハードウェアを用いると、再構成可能ハードウェアの機能の一般性および再利用可能性が犠牲になる。また、ハードウェアで各機能に認められた資源を一致させてハードウェアのバランスおよび最高利用性を与えることは困難であろう。再構成可能I P Pハードウェアを用いると、構成を切り替えるためのオーバーヘッド・コストが避けられない。3つの機能を通る流れに対してサンプル毎に操作すると、最大数のかかる切替えスイッチが必要になるであろう。このシナリオは明らかに最適ではない。このように、再構成の前に各機能をデータのブロックに作用させて機能を切り替えることは、このオーバーヘッドを削減する。こうすれば、各機能に要する時間の量を選択することによって機能間に資源を割り当てることは比較的容易であろう。最後に、かかるブロック処理は、一般に、サンプル・レベルで機能を切り替えることよりもデジタル信号プロセッサ・コアからの制御オーバーヘッドを必要としない。

【0024】種々の機能A、B、C用に選択されたブロックサイズは、必要な相対的データ速度とデータサイズとに依存する。また、デジタル信号プロセッサ・コア110に割り当てられたタスクおよびそれらのそれぞれの計算要件も考慮されなければならない。理想的には、デジタル信号プロセッサ・コア110も再構成可能I P Pハードウェア・コプロセッサ140もほぼフルにロードされる。これは資源の最適利用をもたらす。I P Pに割り当てべき作業量は、I P Pコプロセッサ140対DSPコア110の加速比率に依存する。例えば、I P PがDSPよりも4倍速いときは、最適作業負荷はI P Pに作業の80%を割り当ててDSPに作業の20%を割り当てることであり、5倍の合計スピードアップを達成する。かかるバランスのとれたローディングは、固定された既知の機能と安定なデータ速度とを持つ積アルゴリズムによってだけ達成される。これはほとんどの画像およびビデオ応用の場合であるべきである。計算負荷が時間とともに変わることが予想される場合には、デジタル信号プロセッサ・コア110と再構成可能I P Pハードウェア・コプロセッサ140との間に計算資源を動的に割り当てるのが最もよいと思われる。この場合には、再構成可能I P Pハードウェア・コプロセッサ140によって行われる機能を比較的安定に保ち、ディジタ

ル信号プロセッサ・コア110によって行われる機能だけを変えることが最もよい。

【0025】再構成可能IPPハードウェア・コプロセッサ140の命令集合は、好ましくは、制御機能用のい

```
Receive_data_synchronization (signal, true/false), or wait_until_signal
Send_data_synchronization (signal, true/false), or assert_signal
Synchronization_completion (signal, true/false), or assert_signal
Call_subroutine(subroutine_addr)
Return()
Reset()
Sleep()
Write_parameter(parameter, value)
```

【0027】これらの制御機能は デジタル信号プロセッサ・コア110と再構成可能IPPハードウェア・コプロセッサ140との間の協働において有用である。これらの命令の1番目は受信データ同期命令 (receive\_data\_synchronization command) である。この命令は、信号待ち命令 (wait\_until\_signal command) と呼ぶこともできる。この命令は、一般に、直接メモリ・アクセス回路120によって処理されるデータ転送に関連して用いられるであろう。デジタル信号プロセッサ・コア110は 直接メモリ・アクセス回路120を介して入力データ転送をセットアップすることにより処理を制御する。デジタル信号プロセッサ・コア110は再構成可能IPPハードウェア・コプロセッサ140に2つの命令を送る。第1の命令は受信データ同期命令であり、第2の命令は必要な計算命令である。

【0028】再構成可能IPPハードウェア・コプロセッサ140は、命令待ち行列141に記憶されている命令を先入れ先出し方式で操作する。受信データ同期命令に到達すると 再構成可能IPPハードウェア・コプロセッサは停止するであろう。再構成可能IPPハードウェア・コプロセッサは 入力データ転送の完了を示す指示制御信号を直接メモリ・アクセス回路120から受けるまでアイドル状態にあるであろう。直接メモリ・アクセス回路120は複数の待機データ転送を処理することができることに留意すべきである。これは複数のDMAチャンネルとして当業者に知られている。この場合には、受信データ同期命令は、入力データ転送に用いられるDMAチャンネルに対応するハードウェア信号を指定しなければならない。

【0029】受信データ同期命令が完了すると、再構成可能IPPハードウェア・コプロセッサ140は命令待ち行列141の次の命令に進む。この場合 この次の命令は、たった今ロードされたデータを用いる計算命令である。この計算命令は前の受信データ同期命令が完了するまでは開始することができないので、これは、正しいデータがロードされたことを保証する。

くつかの非計算命令を含む。

【0026】

【表2】

【0030】受信データ同期命令と計算命令との結合は、デジタル信号プロセッサ・コア110の制御負荷を軽減する。デジタル信号プロセッサ・コア110は、直接メモリ・アクセス回路120をセットアップして、入力データ転送を行わせるとともに 命令の対を再構成可能IPPハードウェア・コプロセッサ140に送らせるだけでよい。このことは、計算操作が始まる前に入力データ転送が完了したことを保証する。このことは、再構成可能IPPハードウェア・コプロセッサ140の機能を制御するのにデジタル信号プロセッサ・コア110によって要求されるソフトウェア・オーバーヘッドの量を大幅に削減する。さもないと、デジタル信号プロセッサ・コア110は、入力データ・ロード操作の完了を知らせる割込みを直接メモリ・アクセス回路120から受ける必要がある。割込みを行うには、割込みサービス・ルーチンを立ち上げなければならない。また、かかる割込みは、割込みを受けた処理から割込みサービス・ルーチンへの環境切替えと、割込みから復帰する他の環境切替えとを必要とする。したがって、受信データ同期命令は、デジタル信号プロセッサ・コア内のかなりの容量を生産性の高い用途に向けることができる。

【0031】もう1つの非計算命令は送信データ同期命令である。送信データ同期命令は、ほとんど受信データ同期命令の逆であり、指定された信号を実際に表明する。送信データ同期命令に到達すると、再構成可能IPPハードウェア・コプロセッサ140は、直接メモリ・アクセス動作をトリガする信号を表明する。この直接メモリ・アクセス動作は、別のシステム位置に記憶するためにデータ・メモリ145からデータを読み取る。この直接メモリ・アクセス動作は デジタル信号プロセッサ・コア110によって事前に設定され、送信データ同期命令に出会った後に再構成可能IPPハードウェア・コプロセッサ140から信号を受けると単に開始される。直接メモリ・アクセス回路120が複数のDMAチャンネルをサポートする場合には、送信データ同期命令

は、出力データ転送用の正しいDMAチャンネルをトリガするハードウェア信号を指定しなければならない。または、送信データ同期命令は2つ以上のチャンネルがサポートされている場合にはDMAチャンネルを含めて、直接メモリ・アクセス回路120用の制御パラメータを指定する。かかる送信データ同期命令に出会うと、再構成可能IPPハードウェア・コプロセッサ140は、直接メモリ・アクセス回路120と直接通信して、該当する直接メモリ・アクセス動作をセットアップし開始する。

【0032】別の可能な非計算命令は、同期完了命令、実際には信号表明(assert\_signal)命令の別の応用である。同期完了命令に出会うと、再構成可能IPPハードウェア・コプロセッサ140はデジタル信号プロセッサ・コア110に1つの信号を送る。この信号を受けると、デジタル信号プロセッサ・コア110は、再構成可能IPPハードウェア・コプロセッサ140に送られた全ての以前の命令が完了したことを確認する。応用によっては、割り込みを介してまたはDSPコア110がハードウェア状態レジスタをポーリングすることによってこの信号を検知する方がよい。また、送信および受信データ同期命令を用いて再構成可能IPPハードウェア・コプロセッサ140のいくつかの動作を待ち行列に入れ、待ち行列が終わったときにデジタル信号プロセッサ・コア110に割り込む方がよい場合もある。これは、再構成可能IPPハードウェア・コプロセッサ140による待ち行列動作に続くデジタル信号プロセッサ・コア110による高レベル制御機能に対して有用である。IPPはまた、スリープ・リセット・パラメータ書き込み(Write\_parameter)などの続く他の制御/同期命令を用いる。パラメータ書き込み(write\_parameter)命令はパラメータ更新を行うのに用いる。頻繁に変更されるパラメータは、各タスクで指定される命令に挿入されてもよい。頻繁に変更されないパラメータである、出力右シフト・丸め用の追加項目・飽和の上/下限・飽和の上/下限設定値およびオペランド・サイズ(8/16ビット)のようなパラメータは、パラメータ書き込み(write\_parameter)命令を用いて更新することができる。

【0033】再構成可能IPPハードウェア・コプロセッサは次の計算命令を直接サポートする。

- ・行/列8点DCT/IDCT
- ・ベクトル加算/減算/乗算
- ・スカラー・ベクトル加算/減算/乗算
- ・テーブル・ルックアップ
- ・絶対差分の合計

【0034】また、上の一般的な計算命令の拡張および特殊ケース化により、IPPは次もサポートする。

- ・2-DのDCT/IDCT
- ・簡単な補間によるデモザイク化
- ・色サブサンプリング

・ウェーブレット分析および再構築

・色抑制

・色変換

・メモリ間移動

【0035】各命令は、関係するデータや係数の記憶(入力データ)用のポインタと、結果データを出力するアドレスとを含む。また、フィルタ・タップの数・アップ/ダウン・サンプリング係数・生成された出力の数種々のポインタ増分オプションが計算命令に付加される。画像処理は応用領域であるので、可能であれば2-Dブロック処理が許される。

【0036】図4は、回路100の別の可能な配列を示す。図4に示す回路100は、2個の再構成可能IPPハードウェア・コプロセッサ140・180を含む。デジタル信号プロセッサ・コアは、第1の再構成可能IPPハードウェア・コプロセッサ140および第2の再構成可能IPPハードウェア・コプロセッサ180と共に動作する。専用バス185は、第1の再構成可能IPPハードウェア・コプロセッサ140を第2の再構成可能IPPハードウェア・コプロセッサ180に結合する。これらのコプロセッサは、デジタル信号プロセッサ・コア110のメモリ空間を共用する専用メモリを有する。他方のコプロセッサのメモリによって含まれるアドレス領域に専用バス185を介して一方のコプロセッサから書き込むことによって、データを転送することができる。または、各コプロセッサは、専用バス185に含まれるコプロセッサ間のリンクにより他方のコプロセッサの入力ポートに向けられる出力ポートを有する。この構造は、一方のコプロセッサによって処理される1つの型の動作から第2のコプロセッサによって処理される別の型の動作にデータが流れる積に対して特に有用である。この専用バスを用いることにより、デジタル信号プロセッサ・コア110は直接または直接メモリ・アクセス回路120を介してデータ・ハンドオフを処理しなければならないことから解放される。

【0037】別の例として、図5は、システム・バス142を介して離れて接続されたデジタル信号プロセッサ・コア110および再構成可能IPPハードウェア・コプロセッサ140を示す。デジタル信号プロセッサ・コア110は従来技術の設計のものでよい。好ましい実施の形態では、再構成可能IPPハードウェア・コプロセッサ140は、直接メモリ・アクセス回路120と協働して、デジタル信号プロセッサ・コア110とは独立に自動データ転送を行う。図5に示す構成要素は他の従来技術の構成要素を用いることを排除する意図ではない。図5のシステム・レベル接続は、例えばカタログ装置を用いるときのような、ある構成のデジタル信号プロセッサ・コア140がその内部バスに接続しないときに、有用である。IPPコプロセッサ140がシステム・バスに接続されると、データ転送オーバーヘッドは



通常大きい。同じシステムで多重DSPまたは多重IPPを用いるとかDSPやIPPの変更または更新が比較的容易になるなど、システム・レベルの柔軟度が高まる。

【0038】DSPとIPPとの間の通信の一例として、ベクトル加算タスクを行うようにDSPがIPPに指示した場合には、DSPから見て起こる事象がいくつかある。DSPはDMA転送をセットアップしてデータをIPPに送る。次に、DSPは信号待ち(wait\_until\_signal)命令をIPPに送る(転送が完了するとこの信号はDMAコントローラによって表明されるであろう)。次に、DSPがベクトル加算(vector\_add)命令をIPPに送り、これにより、DSPは解放されて他のタスクを行う。後で、DSPが戻ってきてIPPの完了状態をチェックしてもよいし、ベクトル加算(vector\_add)命令に続くであろう信号表明(assert\_signal)命令を受けIPPタスクの完了時にDSPが割り込みを受けてもよい。最後に、DSPはDMAをセットアップ

してIPPから結果を得る。前に述べたように、各データ転送および各計算命令を管理するのにいくつかのオーバーヘッドがあるので、IPPの機能はブロック計算をサポートし促進する。別の推奨したい方法は、データの同じバッチに対してIPPでカスケードされたタスクを行い、データ転送を減らして、それにより、DSP負荷とシステム・バス負荷と全体の電力消費とを減らすことである。

【0039】IPPは、データが行で記憶されているときに、一次元の行方向遅延をサポートする。アップサンプリングおよびダウンサンプリングのある組合せも同様にサポートされる。例えば、下記の5つの方法は種々のアップ/ダウン・サンプリング・オプションとフィルタ長さの制限とを実現する。ここでは構成A-D(図8および図12)だけを検討する。完全再構成可能IPPデータ経路(図13)にはもっと多くの方法がある。

【0040】

【表3】

方法	a) アップ ダウン・ サンプリング なし	b) $u/s$ 空間時間で アップ サンプル	c) 空間C アップ サンプリング	d) 空間で ダウン・ サンプリング	e) 空間時間で アップ サンプリング
構成	A (8 MAC)	A (8 MAC)	A (8 MAC)	D (カッド 2-ツリー)	D (カッド 2-ツリー)
フィルタ・タップ ( $U+1=1$ )	任意	任意	任意	偶数	偶数
アップサンプリング 係数	1	8, 16, 24	2, 4, 8	1	4, 8, 12
ダウンサンプリング 係数	1	任意	1	2	任意

【0041】図6から図15は、例示の再構成可能IPPハードウェア・コプロセッサの構造を示し、ここで、図8と図10から図15とは、種々のデータ経路構成を示す。図6は、本発明の好ましい実施の形態による再構成可能IPPハードウェア・コプロセッサ140の一般的なアーキテクチャの全体ブロック図を示す。ホストのメモリ・マップでは、IPPインターフェースは、係数・データおよびマクロ命令用の大きな連続メモリ・ブロックとして、また、構成・命令待ち行列・ランタイム制御など用の離散制御/状態レジスタとして見えなければならない。構成/命令待ち行列レジスタは、ホストのDSP外部バス上にI/O空間かメモリ・アドレス空間に置かれるのがよい。ハードウェア・ハンドシェイク信号・ソフトウェア・リセットなどのようなIPPの余り頻繁に変更されないパラメータを変更するには、(ホストに関する)多重書き込みアドレスがセットアップされなければならない。命令用の1つの書き込みアドレスは内部命令待ち行列にリンクする。割り込み解除用のいくつか別の書き込みアドレスが、割り込み毎に1つある。命令完了状態の問合せ用の読取りアドレスが少なくとも1つある。

【0042】できれば、データ部はホストのメモリ空間内にマッピングすべきである。アドレス空間に余裕がな

い場合は、アドレス・ポートとデータ・ポートとを分離して、アドレス・ポートへの書き込みが初期アドレスをセットアップし、データ・ポートへの次く読取り/書き込みがIPPデータ・メモリとの間で連続的にデータを転送するようにすべきである。IPPを実現するためには、バッファが外部16/32ビット・バスと内部メモリの128ビット幅との間に必要である。これには、小さなキャッシュを用いるとよい。読取りにリード・アヘッド方式を用い書き込みにライト・バック方式を用いると、アクセス時間が減る。このバッファの512ビットのうち半分を読取りに半分を書き込みに用いれば、十分である。

【0043】3つの論理メモリ・ブロック、データ・メモリA・Bおよび命令メモリは、外部バス・インターフェースを介してシステム・バスからアクセスすることができる。メモリ・インターフェースは、IPP140とシステム・バス142との間のメモリ調停とシステム・バス・アクセス幅をメモリ幅と一致させるのに必要な簡単な先入れ先出し(FIFO)制御とを処理する。データA・Bは入力/出力データおよび係数用である。カスケードされた命令はデータ・メモリの領域を再使用することができるので、用語「入力/出力」は単一命令の文脈にある。前に述べたように、命令待ち行列141

は、デジタル信号プロセッサ・バス142を介してデジタル信号プロセッサ・コア110から命令を受けて、これらの命令を実行制御ユニット190に与えて、再構成可能IPPハードウェア・コプロセッサ140の動作を制御する。制御ブロックは、所望のメモリ・アクセスと命令によって指示される計算機能とを順に処理する。命令メモリ141は復号ユニット142によって読み取られる。メモリを節約するために、可変長命令が用いられる。復号ユニット142は、生成された制御パラメータ（各命令に1組）を実行制御ユニット190に送る。実行制御ユニット190は、制御パラメータを用いてパイプライン制御経路を駆動して、制御信号を該当する構成要素にファンアウトする。制御信号は、命令内で固定されたものか時間とともに変化するものである。それらはメモリ・アクセス要求、入力／出力フォーマット制御およびデータ経路制御を含む。

【0044】データ・メモリ145と係数メモリ147とは、広いメモリ・ブロック（各128ビット）であり、8方向並列16ビット・データ経路をサポートする。この128ビットの広いメモリ・ブロックを用いることにより、データ経路はサイクル毎にメモリにアクセスする必要がなくなる。データ・メモリ145は、DSPバスを介して関係入力データを受けるとともに、データ経路コア170を通して処理し出力フォーマット180で再書式化した結果データを記憶する。係数データは、DSPバス142から受けるか、恐らくはIPP自身内のルックアップ・テーブルで得られ、入力データと共にデータ経路コア170を通して処理され、出力フォーマット・ブロック180で再書式化される。データ・メモリ145と係数メモリ147とは128ビット語で書き込まれる。この書き込み動作は、命令のオペランド・ポインタを用いて2つのメモリ・ブロックを管理するデジタル信号プロセッサ・コア110または直接メモリ・アクセス回路120によって制御される。アドレス発生器150は、コプロセッサによって用いられるデータおよび係数のリコール用のアドレスを生成する。この読取り動作は各メモリからの128ビットのデータ語で動作する。

【0045】データ・メモリおよび係数メモリからのリコールされた128ビット・データ語は、入力フォーマット160に与えられる。入力フォーマット160は、種々のシフトおよび整列動作を一般的に行って、128ビット入力データ語を所望の計算に必要な順序に配列する。入力フォーマットは、128ビット（ $8 \times 16$ ビット）データAと128ビット（ $8 \times 16$ ビット）データBと128ビット（ $8 \times 16$ ビット）係数データとを出力する。

【0046】これらの3つのデータ・ストリーム、データAとデータBと係数データとは、データ経路170に与えられる。データ経路170はコプロセッサの演算部

である。データ経路は、実行時に形成されて種々の画像処理タスクをサポートする。図12と図13とは本発明の2つの好ましい実施の形態を示す。いくつかのタスクが両方の構成にマッピングされ、それぞれ入力／出力メモリ・アクセスの異なるパターンを与えることができる。これらの選択は、速度とデータ・メモリと場合によってはパワー要求とをバランスするのにアプリケーション・プログラマの手に柔軟を提供する。後で更に説明するように、データ経路170は、種々の方法で接続可能であり種々の乗算・累算動作を行う複数のハードウェア乗算器および加算器を含む。データ経路170は3つの加算器データ・ストリームを出力する。これらの3つのうちの2つは16ビット・データ語であり、その3つのうちの1つは128ビット語（ $8 \times 16$ ビット）である。

【0047】これらの3つのデータ・ストリームは出力フォーマット180の入力に供給される。出力フォーマット180は、メモリに書き戻されるために3つのデータ・ストリームを8つの128ビット・データ語に再配列する。これらの2つの書き込み動作のアドレスはアドレス発生器150によって計算される。この再配列はメモリ語境界上の位置合わせを扱う。コプロセッサの動作は制御ユニット190の制御下にある。制御ユニット190は、命令待ち行列141から命令をリコールするとともに、コプロセッサ140内で対応する制御を行う。

【0048】入力フォーマット160の構造が図7に示されている。それぞれ128ビットの2つのデータ・ストリーム、データAおよびデータBが、マルチプレクサ205・207の入力にそれぞれ供給される。各マルチプレクサは、独立に、1つの入力を選択して、その対応するレジスタ215・217にそれぞれ記憶する。マルチプレクサ205は、入力データ・ストリームの一方を選択するか、レジスタ215の内容を再循環する。マルチプレクサ201は、レジスタ215の内容を選択するか、そのレジスタ211の内容を再循環する。マルチプレクサ207は、入力データ・ストリームの他方を選択するか、レジスタ217の内容を再循環する。シフト221の下位ビットはレジスタ215から与えられる。シフト221の上位ビットはレジスタ211によって与えられる。シフト221はその入力の全256ビットをシフトして選択し、128ビットは全4方向64b $\times$ 2-1マルチプレクサ231に供給され、128ビットは全1方向4方向の128b $\times$ 3-1マルチプレクサ235に供給される。マルチプレクサ231の128ビット出力は、レジスタ241に一時的に記憶されるとともに、データ経路170へのデータA入力を形成する。マルチプレクサ235の128ビット出力は、レジスタ245に一時的に記憶されるとともに、データ経路170へのデータB入力を形成する。マルチプレクサ207の出力は、全1w/2w/4w128b $\times$ 4-1



マルチプレクサ237に直接供給されるとともにレジスタ217に供給される。マルチプレクサ237は、レジスタ217から供給される全128ビットを選択するとともに、その結果をレジスタ247に記憶する。この結果はデータ経路170への係数データ入力を形成する。

【0049】前に述べたように、3つのデータ・ストリーム、データAとデータBと係数データとは、処理のためにデータ経路170に供給される。図8は、本発明の第1の好ましい実施の形態によるデータ経路アーキテクチャを示し、ここでは、8個の乗算累算ユニット(MAC)が並列に接続されている(「A」構成)。複数の積の合計が形成される乗算累算動作は、例えば多くのフィルタ・アルゴリズムにおける信号処理に広く用いられている。N個の乗算累算(この例ではN=8)ユニットが並列に動作されてN個の出力点を計算する。この構成は、画像処理で一般的である多数の画素を含む広いメモリ語に達している。加算器の最終行のフィードバック・ループは、アップサンプリングをサポートする累算器の多数のバンクを含む。好ましい実施の形態によれば、各MACは3個の累算器に接続し、制御ユニット190はこれらの累算器用の必要なアドレス指定機構を含む。3という累算器深さが、3×3マトリクス化を含む色変換をサポートするために選択される。このように、3という累算器深さは色変換の実現を簡単化する。

【0050】図9は、図6に示した出力フォーマッタ180の構造を示す。再構成可能IPPハードウェア・コプロセッサ140内の第1および第2の累算器の16ビット・データ語出力(Acc[0]およびAcc[1])は出力フォーマッタ180への最初の2つの入力を形成し、再構成可能IPPハードウェア・コプロセッサ140の8個の累算器の全ての出力(Acc[0] Acc[1] Acc[2] Acc[3] Acc[4] Acc[5] Acc[6] およびAcc[7])は出力フォーマッタへの第3の入力を提供する。8つの16ビット・ブロックは、出力フォーマッタ180のマルチプレクサおよびレジスタでの処理後にデータ・メモリ145に書き込まれる。

【0051】図10は、単一8ツリー加算器構成(「B」構成)を図示する第2の好ましい実施の形態によるデータ経路170の構造を示す。入力フォーマッタ160から供給されるデータ経路170へのデータAおよびデータBの128ビット(8×16ビット)・データ語入力の種々のセグメントは、加算器/減算器(加算器)310 320 330 340 350 360 370 380に供給される。図に示すように、128ビット出力の最も左のすなわち最上位のビットを表す最初の16ビット・データ語、データA[0]およびデータB[0]は、加算器310および加算器320に結合され、また、第2の16ビット・データ語、データ

A[1]およびデータB[1]は、加算器330および加算器340に結合され、また、第3の16ビット・データ語、データA[2]およびデータB[2]は、加算器350および加算器360に結合され、また、第4の16ビット・データ語データ、A[3]およびデータB[3]は、加算器370および加算器380に結合されている。第1の16ビット・データ語から第4のデータ語のこの加算または減算の結果はパイプライン・レジスタ312 322 332 342 352 362 372 382に記憶される。次に、この結果が係数データと乗算される。IPPのこの構成では、係数データは同じ2つの16ビット・データ語からなる。言い換えると、図10に示した8MAC構成では、4データ語と2係数語とが各サイクルでハードウェアに与えられる。これらの同じ2つの係数語は加算器の各対で用いられ入力データ点と乗算され、パイプライン・レジスタ316 326 336 346 356 366 376 386に記憶されるその積は加算器318 338 358 378で加算される。これらの加算の結果は加算器328 368で加算され、また、その合計は加算器348で加算される。加算器348の出力は累算器349で累算される。この構成の利点は、入力フォーマッタ160の2つの128ビット語出力を処理するのにたとえ8個の乗算器が必要であっても1個の累算器しか必要でないことである。

【0052】図11は、バタフライ加算器を持つ二重4ツリー構成(「C」構成)を図示する第3の好ましい実施の形態によるデータ経路170の構造を示す。入力フォーマッタ160から供給されたデータ経路170へのデータAおよびデータB 128ビット(8×16ビット)・データ語入力の種々のセグメントは、加算器/減算器(加算器)310 320 330 340 350 360 370 380に供給される。図に示すように、128ビット出力の最も左のすなわち最上位のビットを表す最初の16ビット・データ語、データA[0]およびデータB[0]は、加算器310に結合され、また、第2の16ビット・データ語、データA[1]およびデータB[1]は、加算器320に結合され、また、第3の16ビット・データ語、データA[2]およびデータB[2]は、加算器330に結合され、また、第4の16ビット・データ語、データA[3]およびデータB[3]は、加算器340に結合され、また、第5の16ビット・データ語、データA[4]およびデータB[4]は、加算器350に結合され、また、第6の16ビット・データ語、データA[5]およびデータB[5]は、加算器360に結合され、また、第7の16ビット・データ語、データA[6]およびデータB[6]は、加算器370に結合され、また、第8の16ビット・データ語すなわち入力フォーマッタ160の128ビット出力の最下位ビット、

データA [7] およびデータB [7] は、加算器380に結合されている。第1の16ビット・データ語から第8のデータ語のこの加算または減算の結果はパイプライン・レジスタ312 322 332 342 352 362 372 382に記憶される。次に、この結果はIPPのこの構成では2つの16ビット・データ語からなる係数データと乗算される。言い換えると、図11に示す2MAC構成では、8データ語と2係数語とが各サイクルでハードウェアに与えられる。これらの同じ2つの係数語は各MACユニットの各加算器/乗算器部で用いられて入力データ点と乗算され、パイプライン・レジスタ316 326 336 346 356 366 376 386に記憶されるその積は加算器318 338 358 378で加算される。これらの加算の結果は加算器328 368で加算される。次に、加算器328からの合計が、加算器368の合計から減算器388で減算される。次に、減算器388からの出力が累算器359で累算される。次に、加算器368からの合計が加算器328からの合計に加算器348で加算される。加算器348の出力は累算器349で累算される。この構成の利点は、入力フォーマッタ160の2つの128ビット語出力を処理するのにたとえ8個の乗算器が必要であっても2個の累算器しか必要でないことである。

【0053】図12は、カッド2ツリー加算器構成（「D」構成）が図示された第4の好ましい実施の形態によるデータ経路170の構造を示す。入力フォーマッタ160から供給されたデータ経路170へのデータAおよびデータB 128ビット（ $8 \times 16$ ビット）・データ語入力の種々のセグメントは加算器/減算器（加算器）310 320 330 340 350 360 370 380に供給される。2つの異なる入力データ方式が考えられる。第1の方式は、各サイクルで8データ語と2係数語とをハードウェアに与える。2xのダウンサンプリングが濾波で行われる。MACユニットの各対は、2つの乗算を行うとともに、その積の合計を累算する。第2の方式は、各サイクルで2データ語と8係数語とをハードウェアに与える。この場合も、MACユニットの各対は2つの乗算と1つの加算と1つの累算とを行う。アップサンプリングは、4方向並列方式で、また、オプションであるが各累算器の深さで行われる。

【0054】第1の方式によれば、128ビット出力の最も左のすなわち最上位のビットを表す第1の16ビット・データ語、データA [0] およびデータB [0] は、加算器310に結合され、また、第2の16ビット・データ語、データA [1] およびデータB [1] は、加算器320に結合され、また、第3の16ビット・データ語、データA [2] およびデータB [2] は、加算器330に結合され、また、第4の16ビット・データ語、データA [3] およびデータB [3] は、加算器3

40に結合され、また、第5の16ビット・データ語、データA [4] およびデータB [4] は、加算器350に結合され、また、第6の16ビット・データ語、データA [5] およびデータB [5] は、加算器360に結合され、また、第7の16ビット・データ語、データA [6] およびデータB [6] は、加算器370に結合され、また、第8の16ビット・データ語、データA

[7] およびデータB [7] は、加算器380に結合されている。第1のデータ語から第8のデータ語のこの加算または減算の結果はパイプライン・レジスタ312 322 332 342 352 362 372 382に記憶される。次に、この結果は、IPPのこの構成では2つの16ビット係数語からなる係数データと乗算される。言い換えると、図12に示すカッド2ツリー加算器構成では、8データ語と2係数語とが各サイクルでハードウェアに与えられる。同じ2つの係数語はMACユニットの各対で用いられて入力データ点と乗算され、また、パイプライン・レジスタ318 328 338 348 358 368 378 388に記憶されるその積は加算器318 338 358 378で加算される。次に、加算器318 338 358 378からの合計が累算器319 339 359 379で累算される。この構成の利点は、入力フォーマッタ160の2つの128ビット語出力を処理するのにたとえ8個の乗算器が必要であっても4個の累算器だけしか必要としないことである。

【0055】図13は、4つの構成A B C D（図8 図10 図11および図12）をサポートするのに必要な経路選択および多重化を含むデータ経路170の構造を示す。入力フォーマッタ160から供給されたデータ経路170へのデータAおよびデータB 128ビット（ $8 \times 16$ ビット）・データ語入力の種々のセグメントは加算器/減算器（加算器）310 320 330 340 350 360 370 380に供給される。図に示すように、128ビット出力の最も左のすなわち最上位のビットを表す第1の16ビット・データ語、データA [0] およびデータB [0] は、加算器310に結合され、また、第2の16ビット・データ語、データA [1] およびデータB [1] は、加算器320に結合され、また、第3の16ビット・データ語、データA [2] およびデータB [2] は、加算器330に結合され、また、第4の16ビット・データ語、データA [3] およびデータB [3] は、加算器340に結合され、また、第5の16ビット・データ語、データA [4] およびデータB [4] は、加算器350に結合され、また、第6の16ビット・データ語、データA [5] およびデータB [5] は、加算器360に結合され、また、第7の16ビット・データ語、データA [6] およびデータB [6] は、加算器370に結合され、また、第8の16ビット・データ語、データA

【7】およびデータB【7】は、加算器380に結合されている。第1のデータ語から第8のビットのデータ語のこの加算または減算の結果はパイプライン・レジスタ312 322 332 342 352 362 372 382に記憶される。次に、この結果は、IPPのこの構成では同じ16ビットのデータ語からなる係数データと乗算される。言い換えると、図8および図13に示す8MAC構成では、8データ語と1係数語とが各サイクルでハードウェアに与えられる。この同じ係数語は各MACユニットで用いられて入力データ点と乗算され、またパイプライン・レジスタ316 326 336 346 356 366 376 386に記憶されるその積は加算器318 328 338 348 358 368 378 388で累算される。

【0056】実際に、図13の構成A/B/C/D図の経路選択および多重化に示すように、積は加算器318~388への1つの入力を形成する。加算器318への第2の入力はマルチプレクサ319の出力によって形成される。ここで、マルチプレクサ319は2つの入力を有し、その第1は乗算器324からの積であり、その第2は加算器318の累算された合計である。加算器328は両方の入力をマルチプレクサ325 329から受ける。マルチプレクサ325は乗算器324または加算器318の出力を選択する。マルチプレクサ329は加算器328それ自身からまたは次の加算器338からの累算された結果を選択する。8MAC構成（A 図8）では、加算器318 328の対は乗算器314 324からの積を別々に累算する。カッド2ツリー構成（E 図12）では、加算器318 328の対は、（318によって）積を合計したのち、（328によって）この合計を累算する。

【0057】同様に、加算器対338 348と加算器対358 368と加算器対378 388とはそれぞれ、積の別々の累算または2つの積の合計の累算を実行する。カッド2ツリー構成をサポートする合計された累算の場合には、加算器348 368 388は加算器328と同様に最終累算出力を生成する。

【0058】バタフライを持つ二重4ツリー構成（C）をサポートするためには、マルチプレクサ319 339 359 379は、加算器318 338 358 378が8個の乗算器からの積の隣接対を加算するように選択される。マルチプレクサ325 329は加算器328が加算器318 338の結果を加算して最初の4個の乗算器314 324 334 344からの合計を有するように選択される。同様に、マルチプレクサ365 369は、加算器368が最後の4個の乗算器354 364 374 384からの合計を有するように選択される。次に、加算器328 368におけるこれらの2つの合計が、交差加算/減算動作を行う両方の加算器348 390に送られる。加算器34

8は加算を行い、また、加算器390は減算を行う。次に、加算器348 390の結果が累算のために加算器388 392にそれぞれ送られる。加算器388 392は出力の最終対を生成する。

【0059】単一8ツリー構成（B）をサポートするためには、バタフライを持つ二重4ツリー構成（C）の全てのマルチプレクサ構成が保持される。加算器348は8個の乗算器全てからの合計を有し、また、加算器388は累算結果を有する。加算器392の出力は単に無視される。

【0060】図14は、再構成可能データ経路アーキテクチャの簡略化されたバージョンを示す。この簡略化されたアーキテクチャは図8の並列MACおよび図12のカッド2ツリーの両方をサポートする。図に示すように、図8および図13に示すような分離された加算器および乗算器の代わりに、データA入力およびデータB入力の両方が乗算器および加算器/減算器（加算器）の両方に与えられたのち、加算器または乗算器の出力が乗算/加算/減算ブロック810 820 830 840 850 860 870 880から出力する前に選択される。図14の一对のMACユニットの詳細が図28に示されている。各MACユニットは、2つの入力D\_inp C\_inpについてパイプライン化単一サイクル乗算累算動作を行うことができる。D\_inp \* C\_inpの代わりにD\_inp + C\_inpまたはD\_inp - C\_inpの累算も可能であり、したがって、加算/減算ユニット310は各乗算器314と並列に置かれる。マルチプレクサ610は加算器/減算器310出力または乗算器314出力を選択する。MACユニットの各対の間には、（ANDゲート710によって示される）カッド2ツリー・オプションがあり、結果の対（D\_inp \* / + / - C\_inp）を加算して、累算加算器818に与えられるACC\_inpを生成する。

【0061】図14に示すように、上述した構成の両方が実現される。（乗算器と並列のものを除く）8個の加算器だけがいつでも活動状態であるが、この設計では、12個の物理的加算器が多重化および経路選択のコストを削減するために用いられる。交差経路上のANDゲート710 720 730 740は \* / + / - の結果が加算されるべきかどうかを制御する。図28に示すように、3個の累算器612 614 616が各MACユニットで用いられてアップサンプリングを行う。累算器818は、マルチプレクサ618を介して、（他の入力はACC\_inpである）入力としての3つのうちのどれかまたは丸め用のハーフユニット量RND\_ADDから選択することができる。ACC\_inpについての有効データの最初のサイクルでは、RND\_ADDが選択された入力であるべきである。

【0062】丸めと飽和とは主算術データ経路を通る。

累積された合計にハーフユニット量がすでに加えられているので、丸めは単に右シフトである。図15は、図14に示されたものよりも更に簡便化された図8のバージョンを示す。図15に示す構成は、前の構成において示された8個のMACユニットに対して4個のMACユニットだけを含むとともに、図8～図14に示した事前加算を含まない。図14および図28で示したように、図15は、乗算器314および加算器/減算器（加算器）310の両方に与えられたデータA入力およびデータB入力を示し、加算器および乗算器の出力はマルチプレクサ610 620（図28）で多重化される。事前加算がないので、多重化の後に、マルチプレクサ610 620の出力は累算器818 828 838 848で累算される。図14を参照して前に説明したように、また、図28に示すように、3個の累算器612 614 616はアップサンプリングを行うために各MACで利用可能である。累算器818は、マルチプレクサ618を介して、（他の入力はACC\_inpである）入力としての3つのうちのどれかまたは丸め用のハーフユニット量RND\_ADDを選択することができる。ACC\_inpについての有効データの最初のサイクルでは、RND\_ADDが選択された入力であるべきである。

【0063】図14および図15では、乗算/加算/減算ブロックに絶対差動作を加えることが望ましい場合がある。これは、ビデオ符号化応用における動き推定タスクをスピードアップするであろう。図16は、行濾波のIPP動作を行うのに必要な入力データ書式化を示す。第1のサイクルでは、全8個のMACへのデータA入力は第1の8データ語を含む。各サイクル毎に、MACに与えるのに用いられる入力データ語の窓は、右に1語シフトされる。全8個のMACへのデータB入力は同じ係数語と与えられる。この例では、3タップFIRフィルタが実現されるので、3つの係数語と与えられる。

【0064】図中、 $X_0 \dots X_7$ は第1クロック・サイクル中のMACへの第1のデータA入力を含む。1データ語だけシフトすると、第2のデータA入力は第2クロック・サイクル中では $X_1 \dots X_8$ になる。データA入力は、このように続き、各MACにデータ語の連続シーケンスを供給する。第1のフィルタ係数 $C_0$ は第1サイクルの間に全てのMACにばらまかれる。 $C_1$ は第2サイクルの間に全てのMACにばらまかれ、また、 $C_2$ は第3サイクルの間に全てのMACにばらまかれる。第3のサイクルでは、MACユニットは、正しい出力を累算し、結果をデータ・メモリに書き戻すことができる。データ供給は $X_8 \dots X_{15}$ まで続けられて出力 $Y_8 \dots Y_{15}$ の計算を開始し、係数の供給は循環して $C_0$ に戻る。

【0065】同じ構成を維持して、ハードウェアに8データ語と1係数語とを供給する代わりに8フィルタ・バ

ンクに対して1データ語と8係数語とを与えると、別の出力が得られる。この場合も、各MACは、独立に動作し、同じデータ語をその特定の係数語と乗算し、その積を累算する。アップサンプリングは、8方向並列方式で行われ、また、オプションで各累算器の深さで行われる。図17は、対称行濾波動作を行うのに必要な入力データ書式化を示す。この例では、IPPは3タップ・フィルタを実現するので、第1および第3の係数は同じである。したがって、2つの係数語だけが与えられる。第1サイクルでは、データA入力は第1の8データ語 $X_0 \dots X_7$ を含む。第1のデータB入力はデータ語 $X_2 \dots X_9$ を含む。また、全ての乗算器に供給される第1の係数は $C_0$ である。第2のデータA入力は、右に1語シフトされた第1のデータA入力すなわち $X_1 \dots X_8$ である。第2のデータB入力は同じ8データ語である。係数 $C_1$ は第2サイクルで全ての乗算器に供給される。実際には、IPPは次の計算を行う。第1のMACで、 $C_0 * (X_0 + X_2) + 2 * C_1 * X_1$ 第2のMACで、 $C_0 * (X_1 + X_3) + 2 * C_1 * X_2$ など。望ましいフィルタ係数を $F_0 \dots F_2$ 、ただし $F_0 = F_2$ とする。供給される係数は望ましい係数と次のように関連されるべきである。

$$C_0 = F_0$$

$$C_1 = 0.5 * F_1$$

第2サイクルが終わると、3タップ・フィルタ出力はデータ・メモリにいつでも記憶される。第3サイクルでは、データA入力はデータ語 $X_8 \dots X_{15}$ と与えられる。データB入力は $X_{10} \dots X_{17}$ と与えられ、また、係数は循環して $C_0$ に戻る。

【0066】図18は、列フィルタ動作を行うのにデータがメモリのどこから来るかを示す。計算モデルおよび命令体系は、データが行優先順に記憶されることと内積が列に沿って行われることを除いて、行フィルタの計算モデルおよび命令体系と同様である。効率を最大にするには、データと係数と出力配列とは全て $8 \times 16$ ビット・メモリ語に揃えられなければならない。図18に示すように、この場合には、すでに揃えられたデータはメモリ語から直接データ経路に取り出される。言い換えると、データの入力書式化は必要でない。各係数は、図8および図10から図13に示す並列MAC構成の全8個のMACユニットに与えられる。 $N$ タップ列フィルタは、8出力を生成するのに $N+1$ サイクルかかる。各 $N+1$ サイクルに $N$ 回のメモリ読取りと1回のデータ・メモリ書込みとがある。 $N > 8$ のときは、8サイクル毎に1回の係数メモリ読取りがある。そうでない場合は、初期読取りがあったのち、全ての続く係数が入力フォーマットのレジスタによって供給され、それ以上の読取りは必要でない。係数読取り頻度は、行濾波の場合と同じであり、 $N > 8$ の場合には1読取り/8サイクルであり、そうでない場合には0回である。

【0067】図19は、ビデオ符号化の性能を高めるのに用いられる絶対差分の合計を行うのに必要なIPP構成を示す。図19に示すように、データAはX0 X7を含み、また、データBはY0 Y7を含む。係数語は必要でない。各データA入力と各データB入力との差分が減算器310 320 330 340 350 360 370 380で計算され、また、これらの差分はレジスタ312 322 332 342 352 362 372 382に記憶される。次に、正数を得るために、その差分は、その差分が正か負かに従って乗算器314 324 334 344 354 364 374 384でプラス符号またはマイナス符号と乗算される。これらの積は、レジスタ316 326 336 346 356 366 376 386に記憶されたのち、加算器318 328 358 378で合計される。これらの合計は加算器328 348 368で合計される。次に、加算器348の合計が累算器349で累算される。絶対差分の合計に対しては、我々は8ビットの画素について動作するので、加算器は、16ビット幅なければならない最終累算器を除いて、12ビット幅あればよい。飽和しきい値と丸めパラメータとはレジスタの更に別のバンクから得ることができる。

【0068】図20 図21および図22は、行パスおよび列パスのステップを含む離散サイン/コサイン・デモザイク化のIPP動作を示す。大部分のデジタル・スチル・カメラは、インターリーブされた色情報を生成するイメージャで色フィルタ・アレイを用いる。デモザイク化は、欠けている色成分を利用可能な隣接同色成分から得る処理である。簡単な線形内挿法がしばしば用いられ、それは図20で表される。(境界条件を除く)2つまたは4つの最も近い同色近傍があるか否かに従って、重みは0.5か0.25である。

【0069】3色は別々に処理されるが、赤の処理は青の処理と実質的に同じである。各色は行パスと水平パスとの2パスで処理される。行パスは一般的に図21のグラフで表される。緑/赤の各線から、1つのフル緑線と1つのフル赤線とが生成される。緑成分については、行パス濾波は、2相用の係数(0.5 0.0 0.5)および(0.1 0.0)を持つ2相3タップ・フィルタによって実現される。赤成分については、行パス濾波は、係数(0.1 0.0)および(0.5 0.0 0.5)を持つ同じ2相3タップ・フィルタによって実現される。青/緑の各線は同様に処理されてフル青線とフル緑線とを生成する。

【0070】1行から2色出力行を生成することは、アップサンプリングのようなルーピングを用いて、1つの命令に統合されるべきである。8入力画素を処理するのに6サイクルかかる。6サイクルのグループ毎に、1回のデータ・メモリ読取りと2回のデータ・メモリ書込み

と3回の係数メモリ読取りとがある。デモザイク赤/青成分用の列パスの実行が図22aに示されている。赤色および青色については、2タップ列濾波が用いられる。8入力画素を処理するのに3サイクルかかり、その間には2回のデータ・メモリ読取りと1回のデータ・メモリ書込みとがあり、定常状態係数メモリ読取りはない。

【0071】デモザイク緑成分用の列パスの実行が図22bに示されている。緑色成分については、係数(0.25 0.5 0.25)および(0.1 0.0)を持つ2相3タップ列濾波が用いられる。8入力画素は4サイクルで処理される。4サイクルのグループ毎に3回のデータ・メモリ読取りと1回のデータ・メモリ書込みと0回の係数メモリ読取りとがある。要約すると、11サイクルが8入力画素用のデモザイクの内挿法にかかる。13サイクルの中で、6回のデータ・メモリ読取りと4回のデータ・メモリ書込みと3回の係数メモリ読取りとが行われる。

【0072】図23は、ウエーブレット、行パスのIPP動作を行うための入力データの書式化を示す。画像技術では、例えば、テクスチャ特徴用の前処理段階として、画像の圧縮/伸張および特徴抽出にウエーブレットが用いられる。ウエーブレット動作は、図8および図10から図13に示した並列8MAC構成または図14および図15のより簡単化されたバージョンのどれでも実現され得る。ウエーブレット分析の行パスは、2×アップサンプリング(高周波/低周波バンクを得るため)2×ダウンサンプリング(行濾波として実現される)。

【0073】図24は、ウエーブレット動作の列パス部を行うためにメモリのどこから入力データを得るかを示す。列パスは、2×アップサンプリング 2×ダウンサンプリング 列濾波として処理される。この場合も、データ・係数および出力配列は全て8×16ビット・メモリ語に揃えられるべきである。図18に示すように、データはメモリ語からデータ経路に直接取られる。言い換えると、データの入力書式化は必要でない。各係数は、図8および図10から図13に示した並列MAC構成の全8個のMACユニットまたは図14および図15に示した4個のMACユニットに与えられる。8出力を生成するのにN+1サイクルかかる。ただし、Nはウエーブレット核のフィルタ・タップの数である。各N+1サイクルにN回のメモリ読取りと1回のデータ・メモリ書込みとがある。係数読取り回数は行濾波の場合と同じであり、N>8の場合は1読取り/8サイクル、そうでない場合は0回である。ウエーブレット再構成では、2×アップサンプリング・フィルタで高周波および低周波バンクを別々に処理する。最後に、ベクトル加算を用いて2つのバンクを結合する。

【0074】図25は、行パス書式の間接コサイン変換(IDCT)のIPP動作を示す。図に示すように、行

パスIDCTは全マトリクスベクトル方式で行われる。32回の乗算が各8点変換に用いられる。余り効率的には見えないが、IPPをそのまま応用したものである。図8または図10から図15に示した8MAC構成のいずれもこの動作を行うのに用いられ得るが、図11に示したバタフライを持つ分割加算器ツリーの構成が好ましい。この構成は、変換の対称性を利用して乗算回数を半分に減らすことができる。この場合には、IPPは事後乗算/加算器を用いて交差加算/減算を行う。1入力データ語がサイクル毎に広いメモリ語から取り出され、また、8係数語がサイクル毎に用いられる。各8点変換は処理するのに4サイクルかかる。この4サイクルの間に、1回のデータ・メモリ読取りと1回のデータ・メモリ書込みと4回の係数メモリ読取りとが行われる。再構成のバタフライ段階が（例えば図14および図15において）省略される場合には、全 $8 \times 8$ マトリクス乗算法が用いられなければならない。その結果、8点変換当たり64回の乗算を行い、（IPPの8MACまたは4MACを用いて）各変換を行うのに8サイクルまたは16サイクルかかる。図26は、行パス書式の直接コサイン変換（DCT）のIPP動作を示す。行パスIDCTと同様に、行パスDCTは、IPPの構成に従って32回の乗算または64回の乗算で実現され得る。事前乗算加算器構成を持つ二重4ツリー（図11）が利用可能であるときは、それが用いられるべきである。この場合には、バタフライ段階はディスエーブルされる。各メモリ語からの全8データ語はMACに1つずつ与えられる。係数は同様にして与えられ、各MACに異なる係数を1つ与える。この構成で1つの8点変換を処理するのに4サイクルかかる。（例えば、図14および図15において）事前乗算加算器がないので、各8点変換は64回の乗算が必要であり、IPPのMACの数に従って8サイクルまたは16サイクルかかるであろう。

【0075】図27は、列書式単一命令多重データ（SIMD）のIDCTのIPP動作を示す。累算器を多少変更した図8に示した8MACの並列構成が、変換にお

いて対称性を利用するのに必要である。各MACユニットは8個の累算器を必要とし、また、各累算加算器は両方の入力を8個の累算器から取る必要がある。かかるハードウェア機能では、最初の4サイクルの間に、1つの $4 \times 4$ マトリクスは最初の4点を生成するであろう。次の4サイクルの間に、別の $4 \times 4$ マトリクスは次の4点を生成するであろう。サイクル9-10の間に、累算加算器は交差加算/減算を行い、その出力を結合する。したがって、10サイクルで、1対の出力を得て、16点が生成される。この10サイクルの間に、8回のデータ読取りと2回のデータ書込みと8回の係数読取りとが行われる。ハードウェアを変更しなければ、8点変換当たり64回の乗算を行うので、出力の16点はIPPの8MACバージョンで16サイクルかかり、また、IPPの4MACバージョンで32サイクルかかる。どちらの場合でも、別々のMAC構成が用いられる。

【0076】データ経路構成可能性と入力書式化のオプションとに加えて、効率的な制御およびアドレス生成方式がIPPで用いられる。この方式は、ハードウェア制御の実現コストを減少し、IPP用の使用し易いプログラミング・モデルを提供する。

【0077】全ての計算は、ネストされたループ内で行われるであろう。累算器初期化および書出し用のタイミングは、ループ変数を調整することによって制御されるであろう。初期化は、あるループ変数がそれらの最初の値と一致したときに起こるであろう。書出しは、変数の同じ集合がそれらの最終の値と一致したときに起こるであろう。循環累算器は、累算器を指標付けする最も内側のループ・カウントで指定され得る。入力データ、係数および結果の全てのアドレス増分は、「いつ」および「どれだけ」で指定され得る。また、「いつ」はループ変数に関連する。下記は、これらの概念を示すIPP用の制御構造の骨組みの疑似コードである。

【0078】

【表4】

```

dptr = dptr_init; /* ポインタの初期値 */
cptr = cptr_init;
optr = optr_init;

for (i1=0; i1<=lp1end; i1++) {
  for (i2=0; i2<=lp2end; i2++) {
    for (i3=0; i3<=lp3end; i3++) {
      for (i4=0; i4<=lp4end; i4++) {
        /* メモリ読取りおよび入力書式化 */
        x[0..7] = dptr[0..7];
        /* or dptr[0], dptr[0.1], dptr[0.1.2.3] 分散化 */
        y[0..7] = cptr[0..7];
        /* or cptr[0], cptr[0.1], その他 */

        /* 累算器初期化 */
        if (initialize_acc)
          acc[i4*accmode][0..7] = rnd_add[0..7];

        /* 累算 */
        acc[i4*accmode][0..7] += x[0..7] op y[0..7];

        /* 書戻し */
        if (write_back)
          cptr[0..7] = saturate_round(acc[i4*accmode][0..7]);
          /* または、1, 2または4出力 */

        /* ポインタ更新 */
        dptr += ...;
        cptr += ...;
        optr += ...;
      }
    }
  }
}

```

【0079】累算器初期化(initialize\_acc)条件は、ループ・カウンタ変数の指定された部分集合が最初の値(0)と一致するかどうかで判定される。パラメータacc\_loop\_levelは、なし、i4、i4およびi3、または、i4、i3およびi2がテストされるべきかどうかを示す。ループ・カウンタ変数のこの同じ部分集合がそれらの最終値に対してテストされて書戻し(write\_back)条件を与える。ポインタ更新はループ・カウンタ変数を比較することを含む。例えば、4レベルのループでは、データ・ポインタdptrに最大4組のアドレス変更子を与えることができる。各組は、それらの最終の値と一致しなければならないループ・カウンタ変数の部分集合と、条件が真であるときにdptrが増分されるべき量とからなる。同じ機能が係数ポインタcptrと出力ポインタoptrとに与えられる。

【0080】上述した疑似コードでは、パラメータ書込み(write\_parameters)命令で静的に設定されるパラメータまたはIPP計算命令で符号化されるパラメータが用いられる。これらのパラメータは、ループ・カウンタ変数の最終の値(最初の値は必ず0である)と、accmode(単一/循環累算器)とop(乗算/加算/減算/絶対差分)とacc\_loop\_levelと上述したアドレス変更子とを含む。全てのサポートされた画像/ビデオ機能は、上述した形式で書かれたのち、パラメータを適当に設定

することによってIPP命令に変換され得る。IPP用のソフトウェア開発のタスクがこの方法に続く。

#### 【図面の簡単な説明】

【図1】ディジタル信号プロセッサ・コアと本発明による再構成可能ハードウェア・コプロセッサとの組合せであって、コプロセッサがDSPの内部バスに密接に結合されていることを示す。

【図2】ディジタル信号プロセッサ・コアと本発明の再構成可能ハードウェア・コプロセッサとの間のメモリ・マップ論理結合を示す。

【図3】本発明の再構成可能IPPハードウェア・コプロセッサを用いる方法を示す。

【図4】間に専用バスを持つ2個のコプロセッサを含む図1の組合せの別の実施の形態を示す。

【図5】コプロセッサとそのメモリ・ブロックとがシステム・バスでDSPにルーズに接続されているサブシステムを形成するDSPとIPPコプロセッサとの間の別の接続を示す。

【図6】本発明の好ましい実施の形態によるIPP全体ブロック図アーキテクチャを示す。

【図7】図6に示した再構成可能IPPハードウェア・コプロセッサの入力フォーマットを示す。

【図8】8個の独立MACを持つIPPデータ経路アーキテクチャAの略図を示す。



【図9】図6に示した再構成可能IPPハードウェア・コプロセッサの出力フォーマットを示す。

【図10】本発明の好ましい実施の形態によるIPPの加算器部の別の加算器構成、単一8ツリー加算器のIPPデータ経路アーキテクチャBの図を示す。

【図11】好ましい実施の形態によるIPPの加算器部の別の加算器構成、バタフライを持つ二重4ツリーのIPPデータ経路アーキテクチャCの図を示す。

【図12】好ましい実施の形態によるIPPの加算器部の別の加算器構成、カッド2ツリーであるIPPデータ経路アーキテクチャDの図を示す。

【図13a】図8 図10 図11および図12に示したA/B/C/D構成をサポートするのに必要な経路選択および多重化を含むIPP再構成可能データ経路アーキテクチャの図を示す。

【図13b】図8 図10 図11および図12に示したA/B/C/D構成をサポートするのに必要な経路選択および多重送信を含むIPP再構成可能データ経路アーキテクチャの図を示す。

【図14】事前加算のない前記AおよびD構成(図8および図12)をサポートするIPP再構成可能データ経路アーキテクチャの簡単化された構成の図を示す。

【図15】4MACだけを持ち事前加算のない前記A構成だけをサポートするIPPデータ経路アーキテクチャの別の簡単化された構成の図を示す。

【図16】本発明の好ましい実施の形態に従って3タップFIR行濾波を行うのに必要なデータ経路ブロックへの入力係数の再書式化を示す。

【図17】本発明の好ましい実施の形態に従って3タップ対称FIR行濾波を行うのに必要なデータ経路ブロックへの入力係数の再書式化を示す。

【図18】本発明の好ましい実施の形態に従って3タップFIR列濾波を行うのに必要なメモリのどこから入力係数を読み取りどこに出力係数を書き込むかを示す。

【図19】本発明の好ましい実施の形態に従ってIPP

が絶対差分の合計動作を行うときのツリー加算器を持つデータ経路ブロックの略図を示す。

【図20】デモザイク動作に含まれる赤色および青色対緑色のより小さい密度を示す。

【図21】本発明の好ましい実施の形態に従ってデモザイク動作の行バス部を行うのに必要なデータの再書式化を示す。

【図22a】本発明の好ましい実施の形態に従ってデモザイク動作の列バス部を行うのに必要なデータの再書式化を示す。

【図22b】本発明の好ましい実施の形態に従ってデモザイク動作の列バス部を行うのに必要なデータの再書式化を示す。

【図23】本発明の好ましい実施の形態に従って対称行濾波と同様な行方向のウェーブレット変換を行うのに必要な入力データの再書式化を示す。

【図24】本発明の好ましい実施の形態に従って列濾波と同様な列方向のウェーブレット変換を行うのに必要な入力データの再書式化を示す。

【図25】行方向の逆離散コサイン変換(IDCT)の交差加算および減算を行うのに必要なバタフライ構成を持つ分割加算器ツリー構造(C 図11)の事後乗算加算器を示す。

【図26】行方向の離散コサイン変換(DCT)の交差加算および減算を行うのに必要なバタフライ構成を持つバタフライがディスエーブルされた分割加算器ツリー構造(C 図11)の事前乗算加算器を示す。

【図27】列FIR濾波と同様なSIMD動作モードに用いられる列方向のIDCTおよびDCTを示す。

【図28】構成要素が詳細に図示された図14の2個の8MACユニットを示す。

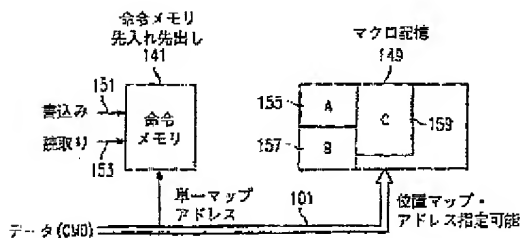
#### 【符号の説明】

110 デジタル信号プロセッサ・コア

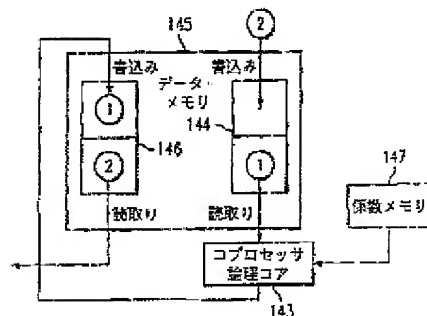
120 直接メモリ・アクセス・コントローラ

140 再構成可能IPPハードウェア・コプロセッサ

【図2】

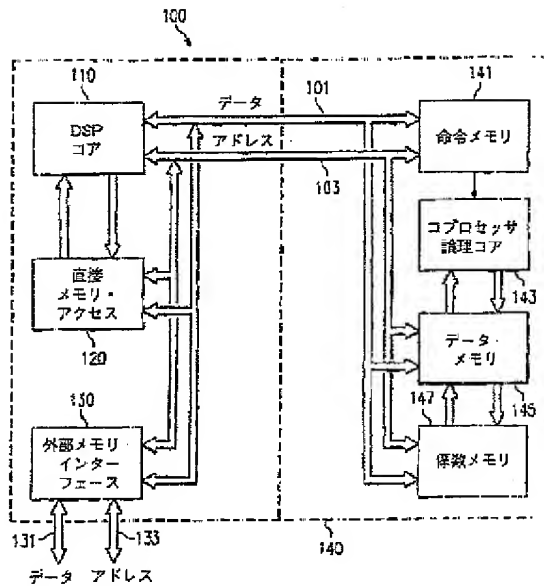


【図3】

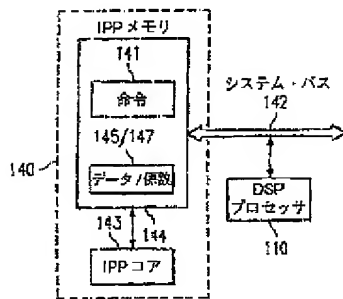




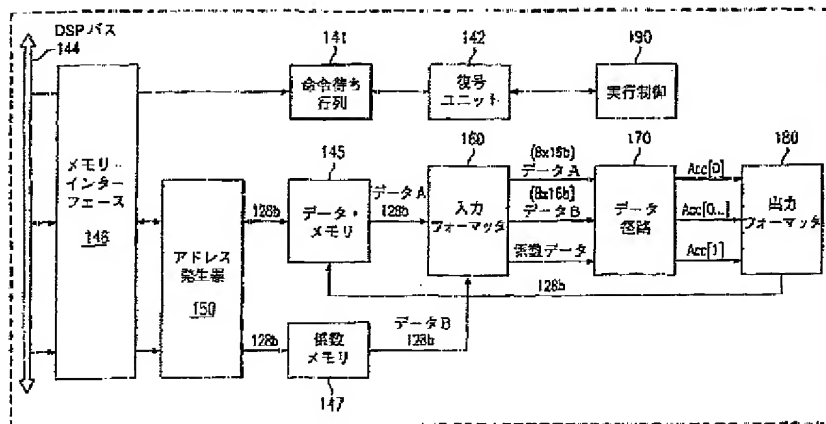
【図1】



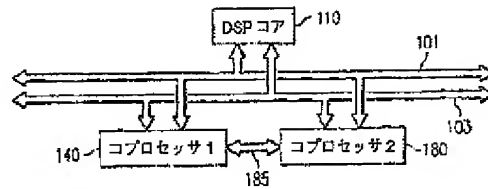
【図5】



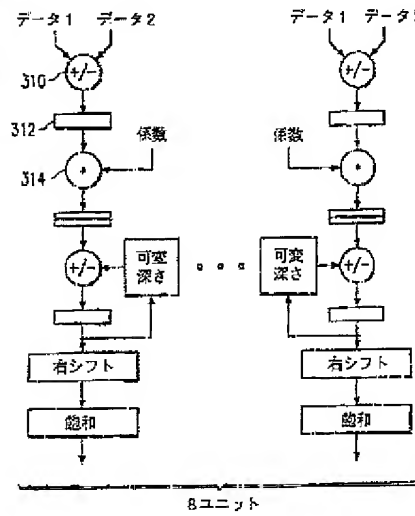
【図6】



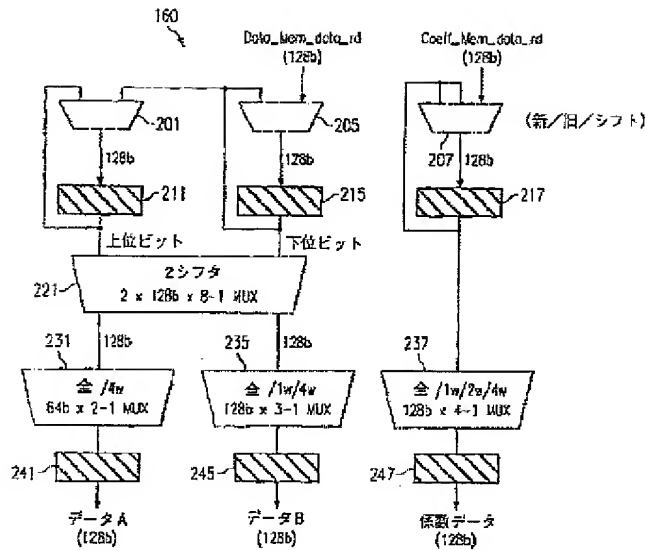
【図4】



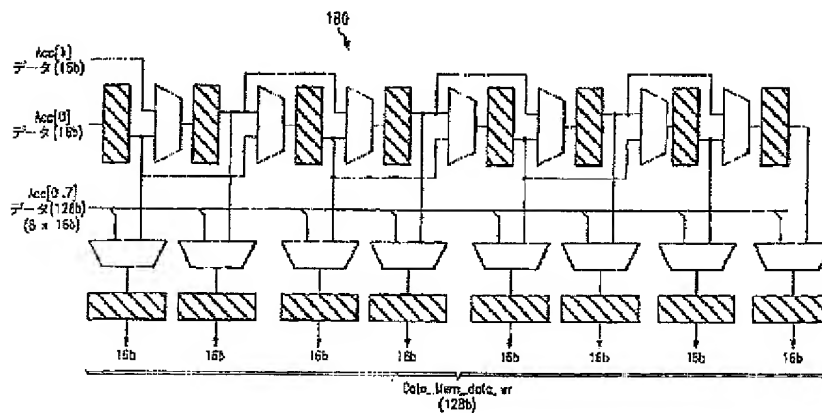
【図8】



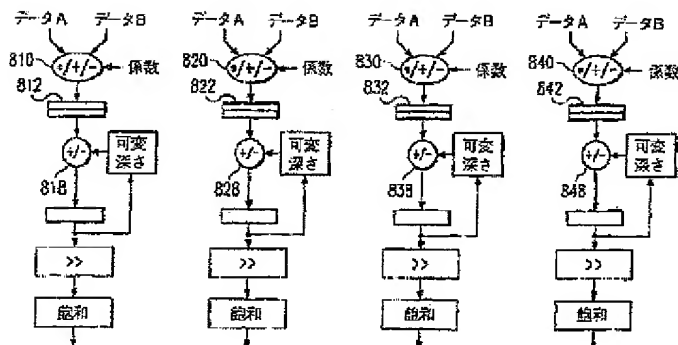
【図7】



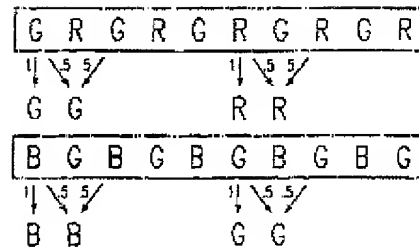
【図9】



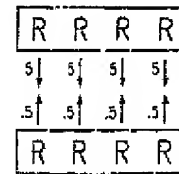
【図15】



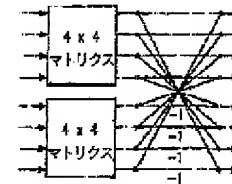
【図21】



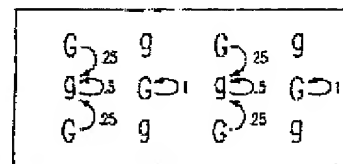
【図22a】



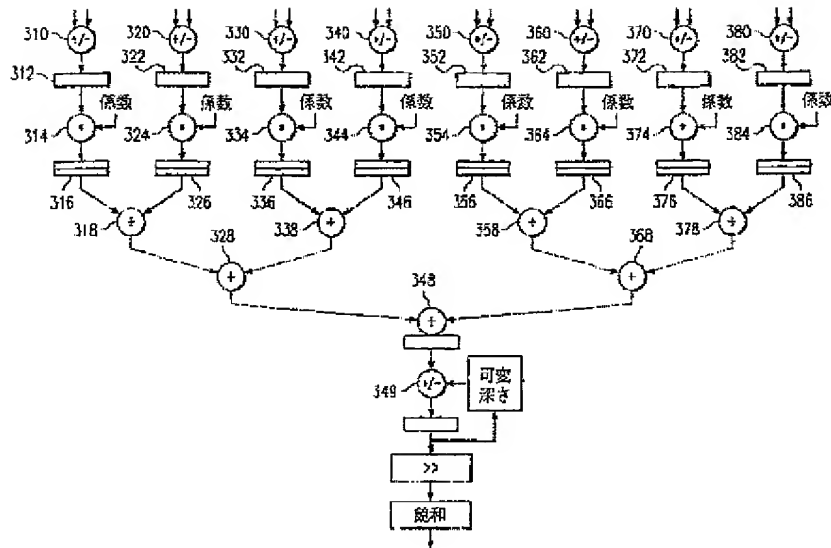
【図25】



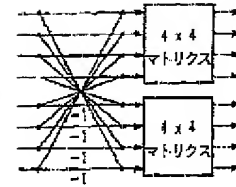
【図22b】



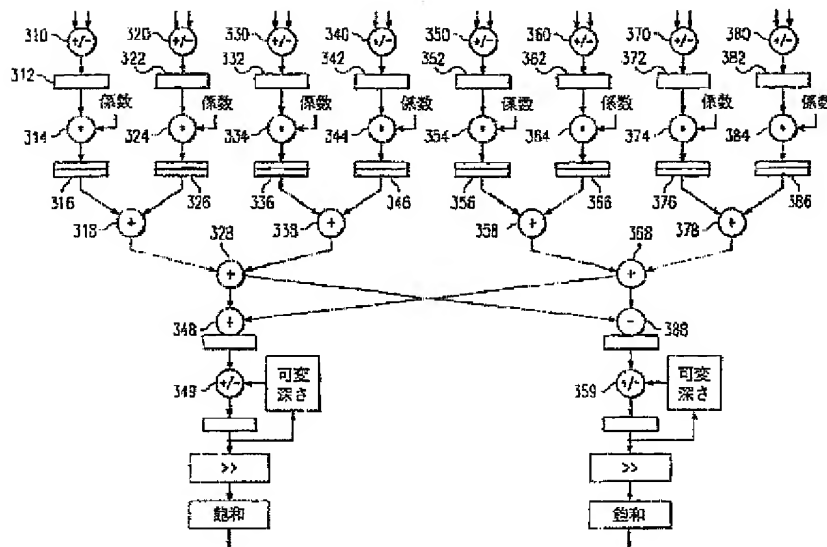
【図 10】



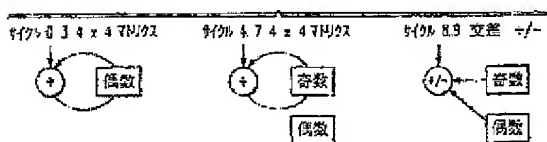
【図 26】



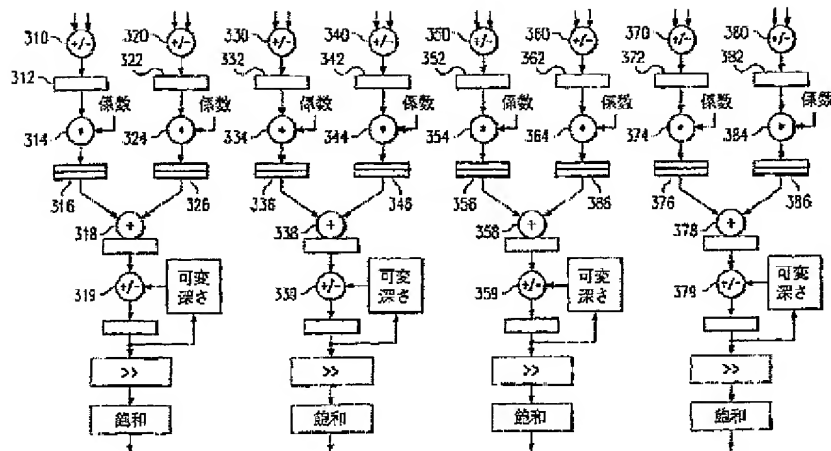
【図 11】



【図 27】

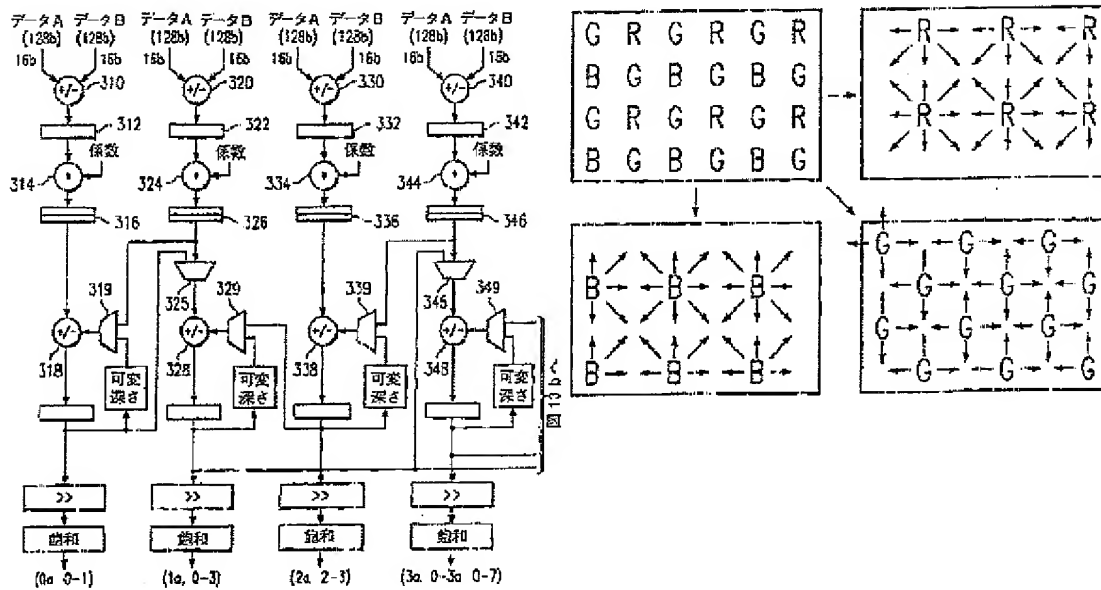


【図12】

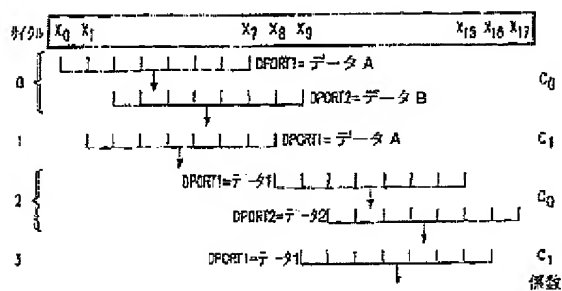


【図13a】

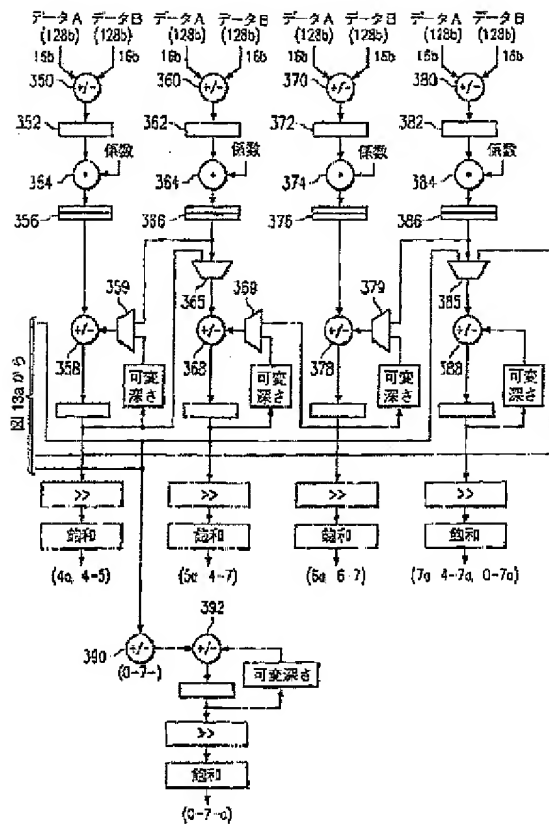
【図20】



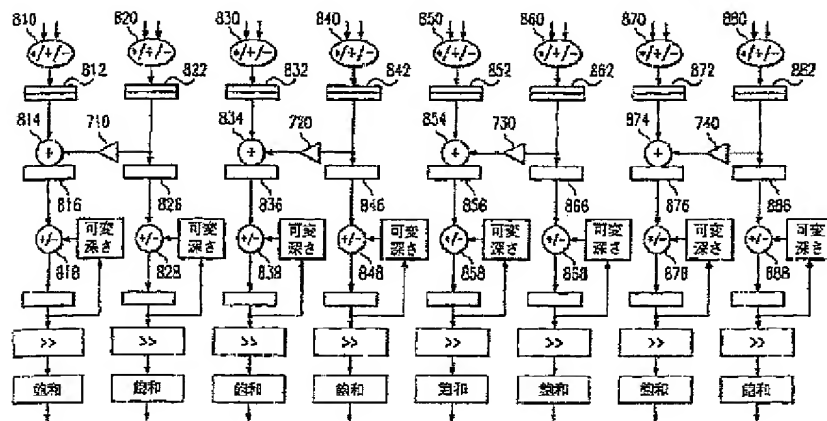
【図17】



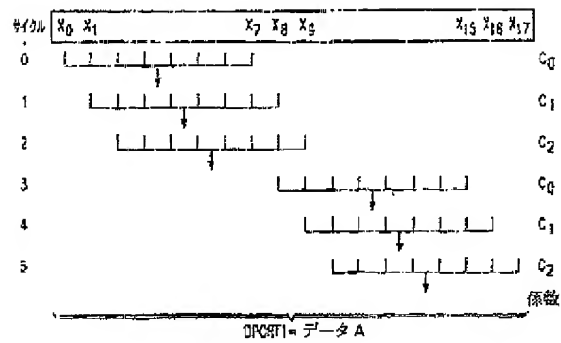
【図13b】



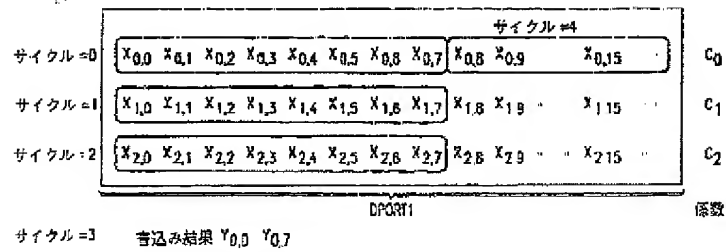
【図14】



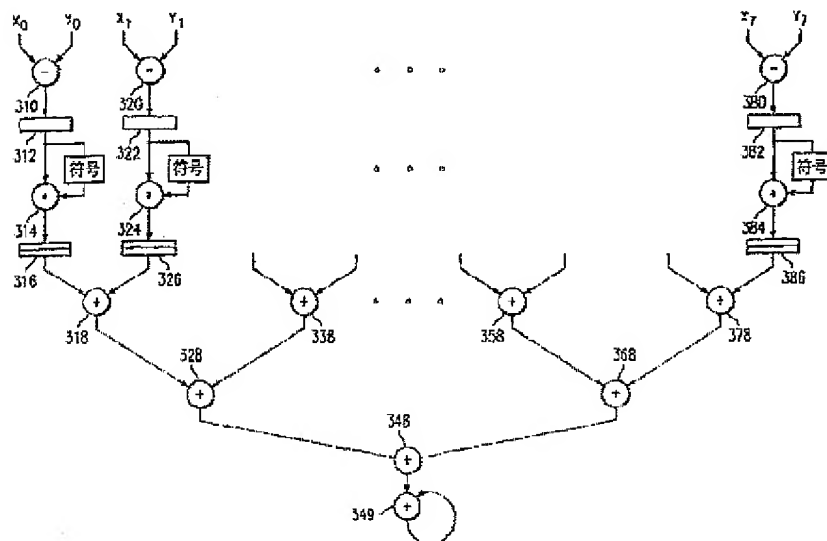
【図16】



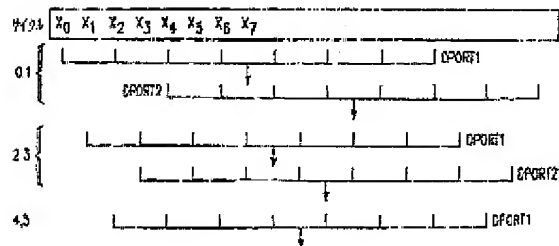
【図18】



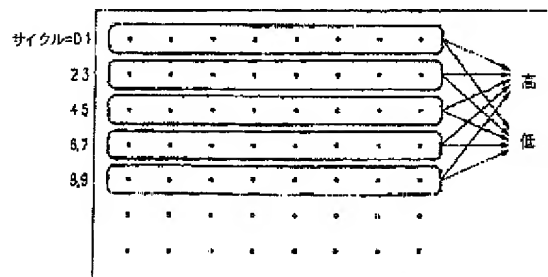
【図19】



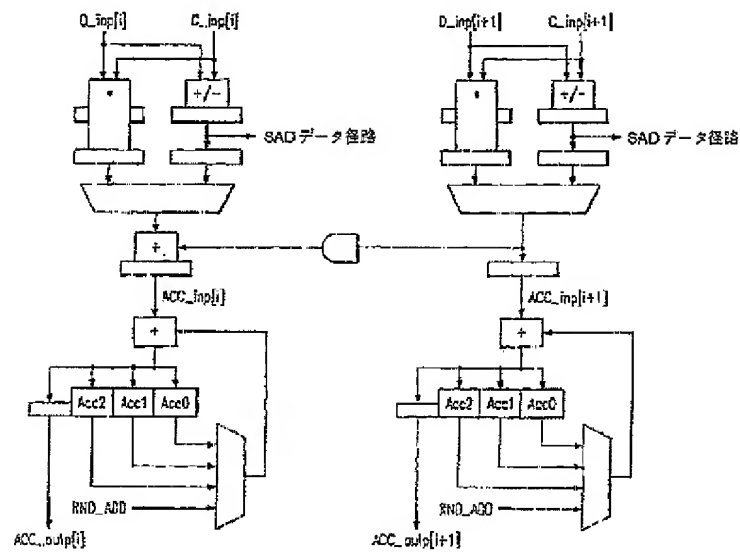
【図23】



【図24】



【図28】



フロントページの続き

(72)発明者 レオナルド ダブリュ・エステベズ  
アメリカ合衆国 テキサス、ダラス、オー  
デリア 12111、 アpartment ナン  
バー 2403

(72)発明者 ウィッサム エイ、ラパディ  
アメリカ合衆国 テキサス、ヒュースト  
ン、エラ リー レーン 9550、アパート  
メント ナンバー 2406



**【外国語明細書】**

- 5    **Reconfigurable SIMD Coprocessor Architecture for Sum of Absolute Differences and Symmetric Filtering(Scalable MAC Engine for Image Processing)**

**Field of the Invention**

- 10    This invention relates in general to signal processing and more specifically to Single Instruction Multiple Data (SIMD) coprocessor architectures providing for faster image and video signal processing, including one and two dimensional filtering, transforms, and other common tasks

**Background of the Invention**

- 15    A problem which has arisen in image processing technology is that two-dimensional (2-D) filtering has a different addressing pattern than one dimensional (1-D) filtering. Previous DSP processors and coprocessors, designed for 1-D, may have to be modified to process 2-D video signals. The end desired goal is to enable a digital signal processor (DSP) or coprocessor to perform image and video processing expediently. In image processing, the most useful  
20    operation is 1-D and 2-D filtering, which requires addressing the 2-D data and 1-D or 2-D convolution coefficients. When the convolution coefficients are symmetrical, architecture that makes use of the symmetry can reduce computation time roughly in half. The primary bottleneck identified for most video encoding algorithms is that of motion estimation. The problem of motion estimation may be addressed by first convolving an image with a kernel to  
25    reduce it into lower resolution images. These images are then reconvolved with the same kernel to produce even lower resolution images. The sum of absolute differences may then be computed within a search window at each level to determine the best matching subimage for a subimage in the previous frame. Once the best match is found at lower resolution, the search is repeated within the corresponding neighborhood at higher resolutions. In view of the above,  
30    a need to produce an architecture capable of performing the 1-D/2-D filtering, preferably symmetrical filtering as well, and the sum of absolute differences with equal efficiency has been generated. Previously, specialized hardware or general purpose DSPs were used to perform the operations of summing of absolute differences and symmetric filtering in SIMD coprocessor architectures. Intel's MMX technology is similar in concept although much more

5     general purpose. Copending applications filed on February 4, 1998, titled "Multi-Multiply-Accumulate (Multi-MAC) coprocessor architecture", Serial No. 60/073,668 (TI-26868) and  
10     "DSP with Efficiently Connected Hardware Coprocessor", Serial No. 60/073,641 (TI-26867) embody host processor/coprocessor interface and efficient Finite Impulse Response/Fast Fourier Transform (FIR/FFT) filtering implementations that this invention is extending to  
15     several other functions

#### Summary of the Invention

The proposed architecture is integrated onto a Digital Signal Processor (DSP) as a coprocessor to assist in the computation of sum of absolute differences, symmetrical  
15     row/column Finite Impulse Response (FIR) filtering with a downsampling (or upsampling) option, row/column Discrete Cosine Transform (DCT)/Inverse Discrete Cosine Transform (IDCT), and generic algebraic functions. The architecture is called IPP, which stands for image processing peripheral, and consists of 8 multiply-accumulate hardware units connected in parallel and routed and multiplexed together. The architecture can be dependent upon a  
20     Direct Memory Access (DMA) controller to retrieve and write back data from/to DSP memory without intervention from the DSP core. The DSP can set up the DMA transfer and IPP/DMA synchronization in advance, then go on its own processing task. Alternatively, the DSP can perform the data transfers and synchronization itself by synchronizing with the IPP architecture on these transfers. This architecture implements 2-D filtering, symmetrical filtering, short  
25     filters, sum of absolute differences, and mosaic decoding more efficiently than the previously disclosed Multi-MAC coprocessor architecture (TI-26868, Serial No. 60/073,641, titled "Reconfigurable Multiple Multiply-Accumulate Hardware Co-Processor Unit", filed on  
January 4, 1998 and incorporated herein by reference). This coprocessor will greatly accelerate the DSP's capacity to perform specifically common 2-D signal processing tasks.  
30     The architecture is also scalable providing an integer speed up in performance for each additional Single Instruction Multiple Data (SIMD) block added to the architecture (provided the DMA can handle data transfers among the DSP and the coprocessors at a rapid enough rate). This technology could greatly accelerate video encoding. This architecture may be

5 integrated onto existing DSPs such as the Texas Instruments TMS320C54x and IMS320C6x  
Each of these processors already contains a DMA controller for data transfers

#### Brief Description of the Drawings

The accompanying drawings, which are incorporated in and constitute a part of the  
10 specification, schematically illustrate a preferred embodiment of the invention and, together  
with the general description given above and the detailed description of the preferred  
embodiment given below, serve to explain the principles of the invention. These and other  
aspects of this invention are illustrated in the drawings, in which:

Figure 1 illustrates the combination of a digital signal processor core and a  
15 reconfigurable hardware co-processor in accordance with this invention, with the coprocessor  
closely coupled to the internal bus of the DSP

Figure 2 illustrates the memory map logical coupling between the digital signal  
processor core and the reconfigurable hardware co-processor of this invention;

Figure 3 illustrates a manner of using the reconfigurable IPP hardware co-processor of  
20 this invention;

Figure 4 illustrates an alternative embodiment of the combination of Fig. 1 including  
two co-processors with a private bus in between;

Figure 5 illustrates an alternate connection between DSP and the IPP coprocessor,  
where the coprocessor and its memory blocks form a subsystem which is loosely connected to  
25 DSP on a system bus

Figure 6 illustrates the IPP overall block diagram architecture according to a preferred  
embodiment of the invention

Figure 7 illustrates the input formatter of the reconfigurable IPP hardware co-processor  
illustrated in Figure 6

30 Figure 8 illustrates a schematic diagram of the IPP Datapath Architecture A, with 8  
independent MACs

Figure 9 illustrates the output formatter of the reconfigurable IPP hardware co-  
processor illustrated in Figure 6

5           Figure 10 illustrates a diagram of the IPP datapath architecture B of one alternative adder configuration of the adder portion of the IPP, the single 8-tree adder, according to a preferred embodiment.

          Figure 11 illustrates a diagram of the IPP datapath architecture C of another alternative adder configurations of the adder portion of the IPP, dual 4-trees with butterfly, according to a  
10 preferred embodiment

          Figure 12 illustrates a diagram of the IPP datapath architecture D of another alternative adder configuration of the adder portion of the IPP, quad-2 trees, according to a preferred embodiment

          Figure 13 illustrates a diagram of the IPP reconfigurable datapath architecture that  
15 includes routing and multiplexing necessary to support the A/B/C/D configurations shown in Figures 8, 10, 11, and 12

          Figure 14 illustrates a diagram of a simplified version of the IPP reconfigurable datapath architecture, which supports the previous A and D version without Pre-Add (Figures 8 and 12).

20           Figure 15 illustrates a diagram of another simplified version of the IPP datapath architecture which only has 4 MACs and supports only the previous A version without Pre-Add

          Figure 16 illustrates the reformatting of the input coefficients to the Datapath block necessary to perform a 3-tap FIR ROW filtering according to a preferred embodiment of the  
25 invention.

          Figure 17 illustrates the reformatting of the input coefficients to the Datapath block necessary to perform a 3 tap symmetric FIR ROW filtering according to a preferred embodiment of the invention

          Figure 18 illustrates from where, in the memory, the input coefficients are read and  
30 whereto the output coefficients are written, necessary to perform a 3-tap FIR column filtering according to a preferred embodiment of the invention

          Figure 19 illustrates a schematic of the data path block with a tree adder when the IPP is performing a sum of absolute differences operation according to a preferred embodiment of the invention

5        Figure 20 illustrates the lesser density of the Red and Blue colors versus the Green color involved in a demosaic operation

      Figure 21 illustrates the reformatting of the data necessary to perform a ROW pass portion of the demosaic operation according to a preferred embodiment of the invention

10       Figure 22 illustrates the reformatting of the data necessary to perform a COLUMN pass portion of the demosaic operation according to a preferred embodiment of the invention

      Figure 23 illustrates the reformatting of the input data necessary to perform row-wise wavelets transform, similar to symmetric ROW filtering, according to a preferred embodiment of the invention

15       Figure 24 illustrates the reformatting of the input data necessary to perform column-wise wavelets transform, similar to column filtering, according to a preferred embodiment of the invention

      Figure 25 illustrates the post-multiplier adders of a split adder tree with butterfly configuration (C, Figure 11) necessary to implement the cross additions and subtractions of the row-wise Inverse Discrete Cosine Transform(IDCT)

20       Figure 26 illustrates the pre-multiplier adders of a split adder tree with butterfly configuration (C, Figure 11) with the butterfly disabled necessary to implement the cross additions and subtractions of the row-wise Discrete Cosine Transform(DCI).

      Figure 27 illustrates the column-wise IDCT and DCT implemented in SIMD mode of operation, similar to the column FIR filtering

25       Figure 28 illustrates two of the 8 MAC units of Figure 14 in a more detailed drawing of components

#### Detailed Description of the Preferred Embodiments

30       Figure 1 illustrates circuit 100 including digital signal processor core 110 and a reconfigurable LPP hardware co-processor 140. Figure 1 is the same figure 1 as in co-pending application Serial No. 60/073,641, titled "Reconfigurable Multiple Multiply-Accumulate Hardware Co-processor Unit" assigned to the same assignee, the co-processor of which a preferred embodiment of this invention is made. In accordance with a preferred embodiment of this invention, these parts are formed in a single integrated circuit (IC). Digital signal

5 processor core 110 may be of convention design. The IPP is a memory mapped peripheral. Transferring data between IPP's and DSP's working memory can be carried out via the Direct Memory Access (DMA) controller 120 without intervention from the digital signal processor core 110. Alternatively, the DSP core 110 can handle data transfer itself via direct load/store to IPP's working memory 141, 145 and 147. A combination of the two transfer mechanisms is  
10 also possible, as the DMA can handle large data/coefficient transfers more efficiently, and the DSP can directly write out short commands to IPP command memory 141 more efficiently.

The reconfigurable IPP hardware co-processor 140 has a wide range of functionality and supports symmetrical/asymmetrical row/column filtering, 2-D filtering, sum of absolute differences, row/column DCI/IDCI and generic linear algebraic functions. Symmetrical  
15 row/column filtering is frequently used in up/down sampling to resize images to fit display devices. Two-dimensional filtering is often used for demosaic and for image enhancement in digital cameras. Sum of absolute differences is implemented in MPEG video encoding and H 263 and H 323, encoding standards for the telephone line video conferencing. Row/column DCT/IDCT is implemented in JPEG image encoding/decoding and MPEG video  
20 encoding/decoding. Generic linear algebraic functions, including array addition/subtraction and scaling are frequently used in imaging and video applications to supplement the filtering and transform operations. For example, digital cameras require scaling of pixels to implement gain control and white balancing.

In the preferred embodiment, reconfigurable IPP hardware co-processor 140 can be  
25 programmed to coordinate with direct memory access circuit 120 for autonomous data transfers independent of digital signal processor core 110. External memory interface 130 serves to interface the internal data bus 101 and address bus 103 to their external counterparts external data bus 131 and external address bus 133, respectively. External memory interface 130 is conventional in construction. Integrated circuit 100 may optionally include additional  
30 conventional features and circuits. Note particularly that the addition of cache memory to integrated circuit 100 could substantially improve performance. The parts illustrated in Figure 1 are not intended to exclude the provision of other conventional parts. Those conventional parts illustrated in Figure 1 are merely the parts most effected by the addition of reconfigurable hardware co-processor 140.

5           Reconfigurable IPP hardware co-processor 140 is coupled to other parts of integrated circuit 100 via a data bus 101 and address bus 103. Reconfigurable IPP hardware co-processor 140 includes command memory 141, co-processor logic core 143, data memory 145, and coefficient memory 147. Command memory 141 serves as the conduit by which digital signal processor core 110 controls the operations of reconfigurable hardware co-processor 140. Co-processor logic core 143 is responsive to commands stored in command memory 141 which form a command queue to perform co-processing functions. These co-processing functions involve exchange of data between co-processor logic core 143 and data memory 145 and coefficient memory 147. Data memory 145 stores the input data processed by reconfigurable hardware co-processor 140 and further stores the resultant of the operations of reconfigurable hardware co-processor 140. Coefficient memory 147 stores the unchanging or relatively unchanging process parameters called coefficients used by co-processor logic core 143. Though data memory 145 and coefficient memory 147 have been shown as separate parts, it would be easy to employ these merely as different portions of a single, unified memory. As will be shown below, for the multiple multiply accumulate co-processor described, it is best if such a single unified memory has two read ports for reading data and coefficients and one write port for writing output data. As multiple-port memory takes up more silicon area than single-port memory of the same capacity, the memory system can be partitioned to blocks to achieve multiple access points. With such memory configuration, it is desirable to equip IPP with memory arbitration and stalling mechanism to deal with memory access conflicts. It is believed best that the memory accessible by reconfigurable IPP hardware co-processor 140 be located on the same integrated circuit in physical proximity to co-processor logic core 143. This physical closeness is needed to accommodate the wide memory buses required by the desired data throughput of co-processor logic core 143.

Figure 2 illustrates the memory mapped interface between digital signal processor core 110 and reconfigurable IPP hardware coprocessor 140. Digital signal processor core 110 controls reconfigurable IPP hardware coprocessor 140 via command memory 141. In the preferred embodiment, command memory 141 is a first-in-first-out (FIFO) memory with a command queue. The write port of command memory 141 is memory mapped into a single memory location within the address space of digital signal processor core 110. Thus digital

5 signal processor core 110 controls reconfigurable IPP hardware co-processor 140 by writing  
commands to the address serving as the input to command memory 141. Command memory  
141 preferably includes two circularly oriented pointers. The write pointer 151 points to the  
location within command memory 141 wherein the next received command is to be stored.  
Each time there is a write to the predetermined address of command memory 141, write  
10 pointer selects the physical location receiving the data. Following such a data write, write  
pointer 151 is updated to point to the next physical location within command memory 141.  
Write pointer 151 is circularly oriented in that it wraps around from the last physical location  
to the first physical location. Reconfigurable IPP hardware co-processor 140 reads commands  
from command memory 141 in the same order as they are received (FIFO) using read pointer  
15 153. Read pointer 153 points to the physical location with command memory 141 storing the  
next command to be read. Read pointer 153 is updated to reference the next physical location  
within command memory 141 following each such read. Note that read pointer 153 is also  
circularly oriented and wraps around from the last physical location to the first physical  
location. Command memory 141 includes a feature preventing write pointer 151 from passing  
20 read pointer 153. This may take place, for example, by refusing to write and sending a  
memory fault signal back to digital signal processor core 110 when write pointer 151 and read  
pointer 153 reference the same physical location. Thus the FIFO buffer of command memory  
141 can be full and not accept additional commands.

Many digital signal processing tasks will use plural instances of similar functions. For  
25 example, the process may include several filter functions. Reconfigurable IPP hardware co-  
processor 140 preferably has sufficient processing capability to perform all of these filter  
functions in real time. The macro store area 149 can be used to store common function in  
form of subroutines so that invoking these functions takes just a "call subroutine" command in  
the command queue 141. This reduces traffic on the command memory and potentially  
30 memory requirement on the command memory as a whole. Figure 2 illustrates 3 subroutines  
A, B, and C residing on the macro store area 149, with each subroutine ending with a "return"  
command.

Alternate to the command FIFO/macro store combination is static command memory  
contents that DSP set up initially. The command memory can hold multiple command



5 sequences, each ending with a "sleep" command. DSP instructs IPP to execute a particular command sequence by writing the starting address of the sequence to an IPP control register. IPP executes the specified commands, until encountering the sleep command, when it goes into standby mode waiting for further instruction from the DSP. Data memory 145 and coefficient memory 147 can both be mapped within the Data address space of digital signal processor core 110. As illustrated in Figure 2, Data bus 101 is bidirectionally coupled to memory 149. In accordance with the alternative embodiment noted above, both data memory 145 and coefficient memory 147 are formed as a part of memory 149. Memory 149 is also accessible by co-processor logic core 143(not illustrated in Figure 2). Figure 2 illustrates three circumscribed areas of memory within memory 149. As will be further described below, 15 reconfigurable hardware co-processor 140 performs several functions employing differing memory areas.

Integrated circuit 100 operates as follows. Either digital signal processor core 110 or DMA controller 120 control the data and coefficients used by reconfigurable IPP hardware co-processor 140 by loading the data into data memory 145 and the coefficients into coefficient 20 memory 147 or, alternatively, both the data and the coefficients into unified memory 149. Digital signal processor core 110 may be programmed to perform this data transfer directly, or alternatively, digital signal processor core 110 may be programmed to control DMA controller 120 to perform this data transfer. Particularly for audio or video processing applications, the data stream is received at a predictable rate and from a predictable device. Thus it would be 25 typically efficient for digital processor core 110 to control DMA controller 120 to make transfers from external memory to memory accessible by reconfigurable hardware co-processor 140.

Following the transfer of data to be processed, digital signal processor core 110 signals reconfigurable IPP hardware co processor core 140 with the command for the desired signal 30 processing algorithm. As previously stated, commands are sent to a reconfigurable IPP hardware co-processor 140 by a memory write to a predetermined address within Command Queue 141. Received commands are stored in Command Queue 141 on a first-in-first-out basis. Each computational command of reconfigurable IPP co-processor preferably includes a manner to specify the particular function to be performed. In the preferred embodiment,

5    reconfigurable hardware co-processor is constructed to be reconfigurable. Reconfigurable IPP  
hardware co-processor has a set of functional units, such as multipliers and adders, that can be  
connected together in differing ways to perform different but related functions. The set of  
related functions selected for each reconfigurable hardware co-processor will be based upon a  
similarity of the mathematics of the functions. This similarity in mathematics enables similar  
10 hardware to be reconfigured for the plural functions. The command may indicate the  
particular computation via an opcode in the manner of data processor instructions.

Each computational command includes a manner of specifying the location of the input  
data to be used by the computation. There are many suitable methods of designating data  
space. For example, the command may specify a starting address and number of data words or  
15 samples within the block. The data size may be specified as a parameter or it may be specified  
by the op code defining the computation type. As a further example, the command may  
specify the data size, the starting address and the ending address of the input data. Note that  
known indirect methods of specifying where the input data is stored may be used. The  
command may include a pointer to a register or a memory location storing any number of these  
20 parameters such as start address, data size, and number of samples within the Data block and  
end address.

Each computational command must further indicate the memory address range storing  
the output data of the particular command. This indication may be made by any of the  
methods listed previously with regard to the locations storing the input data. In many cases the  
25 computational function will be a simple filter function and the amount of output data following  
processing will be about equivalent to the amount of input data. In other cases, the amount of  
output data may be more or less than the amount of input data. In any event, the amount of  
resultant data is known from the amount of input Data and the type of computational function  
requested. Thus merely specifying the starting address provides sufficient information to  
30 indicate where all the resultant data is to be stored. It is feasible to store output data in a  
destructive manner over-writing input data during processing. Alternatively, the output data  
may be written to a different portion of memory and the input data preserved at least  
temporarily. The selection between these alternatives may depend upon whether the input data  
will be reused.

5           Figure 3 illustrates one useful technique involving alternatively employing two memory areas. One memory area 145 stores the input data needed for co-processor function. The relatively constant coefficients are stored in coefficient memory 147. The input data is recalled for use by co-processor logic core 143(1 read) from a first memory area 144 of the data memory 145. The output data is written into the second memory area 146 of the data memory(1 write). Following use of the data memory area, direct memory access circuit 120 writes the data into the first memory area 144 for the next block, overwriting the data previously used (2 write). At the same time, direct memory access circuit 120 reads data from second memory area 146 ahead of it being overwritten by reconfigurable hardware co-processor 140 (2 read). These two memory areas for input Data and for resultant data could be configured as circular buffers. In a product that requires plural related functions, separate memory areas defined as circular buffers can be employed. One memory area configured as a circular buffer will be allocated to each separate function.

20           The format of computational commands preferably closely resembles the format of a subroutine call instruction in a high level language. That is, the command includes a command name similar in function to the subroutine name specifying the particular computational function to be performed. Each command also includes a set of parameters specifying available options within the command type. For example, the following list of computational commands and the various parameters:

25           Row\_filter(us, ds, length, block, data\_addr, coef\_addr, outp\_addr)  
             Column\_filter(us, ds, length, block, data\_addr, coef\_addr, outp\_addr)  
             Row\_filter\_sym(us, ds, length, block, data\_addr, coef\_addr, outp\_addr)  
             Sum\_abs\_diff(length, data\_addr1, data\_addr2, outp\_addr)  
             Row\_DCI(data\_addr, outp\_addr), Row\_IDCT, Column\_DCT, Column\_IDCT  
             Vector\_add(length, data\_addr1, data\_addr2, outp\_addr)

30           These parameters may take the form of direct quantities or variables, which are pointers to registers or memory locations storing the desired quantities. The number and type of these parameters depends upon the command type. This subroutine call format is important in reusing programs written for digital signal processor core 110. Upon use, the programmer or compiler provides a stub subroutine to activate reconfigurable IPP hardware co-processor 140.

5 This stub subroutine merely receives the subroutine parameters and forms the corresponding co-processor command using these parameters. The stub subroutine then writes this command to the predetermined memory address reserved for command transfers to reconfigurable hardware co-processor 140 and then returns. This invention envisions that the computational capacity of digital signal processor cores will increase regularly with time. Thus the  
10 processing requirements of a particular product may require the combination of digital signal processor core 110 and reconfigurable IPP hardware co-processor 140 at one point in time. At a later point in time, the available computational capacity of an instruction set digital signal processor core may increase so that the functions previously requiring a reconfigurable IPP hardware co-processor may be performed in software by the digital signal processor core. The  
15 prior program code for the product may be easily converted to the new, more powerful digital signal processor. This is achieved by providing independent subroutines for each of the commands supported by the replaced reconfigurable hardware co-processor. Then each place where the original program employs the subroutine stub to transmit a command is replaced by the corresponding subroutine call. Extensive reprogramming is thus avoided.

20 Following completion of processing on one block of data, the data may be transferred out of data memory 145. This second transfer can take place either by direct action of digital signal processor core 110 reading the data stored at the output memory locations or through the aid of direct memory access circuit 120. This output data may represent the output of the process. In this event, the data is transferred to a utilization device. Alternatively, the output  
25 data of reconfigurable IPP hardware co-processor 140 may represent work in progress. In this case, the data will typically be temporarily stored in memory external to integrated circuit 100 for later retrieval and further processing.

Reconfigurable IPP hardware co-processor 140 is then ready for further use. This further use may be additional processing of the same function. In this case, the process  
30 described above is repeated on a new block of data in the same way. This further use may be processing of another function. In this case, the new block of data must be loaded into memory accessible by reconfigurable IPP hardware co-processor 140, the new command loaded and then the processed data read for output or further processing.

5           Reconfigurable IPP hardware co-processor 140 preferably will be able to perform more than one function of the product algorithm. The advantage of operating on blocks of data rather than discrete samples will be evident when reconfigurable IPP hardware co-processor 140 operates in such a system. As an example, suppose that reconfigurable IPP hardware co-processor 140 performs three functions, A, B and C. These functions may be sequential or  
10 they may be interleaved with functions performed by digital signal processor core 110. Reconfigurable IPP hardware co-processor 140 first performs function A on a block of data. This function is performed as outlined above. Digital signal processor core 110 either directly or by control of direct memory access circuit 120 loads the input data into data memory 145. Upon issue of the command for configuration for function A which specifies the amount of  
15 data to be processed, reconfigurable IPP hardware co-processor 140 performs function A and stores the resultant data back into the portion of memory 145 specified by the command. A similar process occurs to cause reconfigurable IPP hardware co-processor 140 to perform function B on data stored in memory 145 and return the result to memory 145. The performance of function A may take place upon Data blocks having a size unrelated to the size  
20 of the Data blocks for function B. Finally, reconfigurable IPP hardware co-processor 140 is commanded to perform function C on data within memory 145, returning the resultant to memory 145. The block size for performing function C is independent of the block sizes selected for functions A and B.

          The usefulness of the block processing is seen from this example. The three functions  
25 A, B and C will typically perform amounts of work related to one common data processing size (for example, one 16 x 16 block of pixels as a final output), that is not necessarily equal in actual input/output sizes due to filter history and up/down sampling among functions. Provision of special hardware for each function will sacrifice the generality of functionality and reusability of reconfigurable hardware. Further, it would be difficult to match the resources  
30 granted to each function in hardware to provide a balance and the best utilization of the hardware. When reconfigurable hardware is used there is inevitably an overhead cost for switching between configurations. Operating on a sample by sample basis for flow through the three functions would require a maximum number of such reconfiguration switches. This scenario would clearly be less than optimal. Thus operating each function on a block of Data

5 before reconfiguration to switch between functions would reduce this overhead. Additionally,  
it would then be relatively easy to allocate resources between the functions by selecting the  
amount of time devoted to each function. Lastly, such block processing would generally  
require less control overhead from the digital signal processor core than switching between  
functions at a sample level.

10 The block sizes selected for the various functions A, B and C will depend upon the  
relative data rates required and the data sizes. In addition, the tasks assigned to digital signal  
processor core 110 and their respective computational requirements must also be considered.  
Ideally, both digital signal processor core 110 and reconfigurable IPP hardware co-processor  
140 would be nearly fully loaded. This would result in optimum use of the resources. The  
15 amount of work that should be assigned to the IPP depends on the speedup factor of the IPP  
co-processor 140 versus the DSP core 110. For example, when the IPP is 4 times faster than  
the DSP, the optimum workload is to assign 80% of the work to the IPP, and 20% to the DSP  
to accomplish 5 times the total speedup. Such balanced loading may only be achieved with  
product algorithms with fixed and known functions and a stable data rate. This should be the  
20 case for most imaging and video applications. If the computational load is expected to change  
with time, then it will probably be best to dynamically allocate computational resources  
between digital signal processor core 110 and reconfigurable IPP hardware co-processor 140.  
In this case it is best to keep the functions performed by reconfigurable IPP hardware co-  
processor 140 relatively stable and only the functions performed by digital signal processor  
25 core 110 would vary.

The command set of Reconfigurable IPP hardware co-processor 140 preferably includes  
several non-computational instructions for control functions:

Receive\_data\_synchronization (signal, true/false), or wait\_until\_signal  
Send\_data\_synchronization (signal, true/false), or assert\_signal  
30 Synchronization\_completion (signal, true/false), or assert\_signal  
Call\_subroutine(subroutine\_addr)  
Return()  
Reset()  
Sleep()

5       Write\_parameter(parameter, value)

These control functions will be useful in cooperation between digital signal processor core 110 and reconfigurable IPP hardware co-processor 140. The first of these commands is a receive\_data\_synchronization command. This command can also be called a wait\_until\_signal command. This command will typically be used in conjunction with data transfers handled by direct memory access circuit 120. Digital signal processor core 110 will control the process by setting up the input data transfer through direct memory access circuit 120. Digital signal processor core 110 will send two commands to reconfigurable IPP hardware co-processor 140. The first command is the receive data synchronization command and the second command is the computational command desired.

15       Reconfigurable IPP hardware co-processor 140 operates on commands stored in the command queue 141 on a first-in-first-out basis. Upon reaching the receive data synchronization command, reconfigurable IPP hardware co-processor will stop. Reconfigurable IPP hardware co-processor will remain idle until it receives the indicated control signal from direct memory access circuit 120 indicating completion of the input data transfer. Note that direct memory access circuit 120 may be able to handle plural queued data transfers. This is known in the art as plural DMA channels. In this case, the receive data synchronization command must specify the hardware signal corresponding to the DMA channel used for input data transfer.

Following the completed receive data synchronization command, reconfigurable IPP hardware co-processor 140 advances to the next command in Command Queue 141. In this case, this next command is a computational command using the data just loaded. Since this computational command cannot start until the previous receive data synchronization command completes, this assures that the correct data has been loaded.

25       The combination of the receive data synchronization command and the computational command reduces the control burden on digital signal processor core 110. Digital signal processor core 110 need only set up direct memory access circuit 120 to make the input data transfer and send the pair of commands to reconfigurable IPP hardware co-processor 140. This would assure that the input data transfer had completed prior to beginning the computational operation. This greatly reduces the amount of software overhead required by

5 the digital signal processor core 110 to control the function of reconfigurable IPP hardware co-processor 140. Otherwise, digital signal processor core 110 may need to receive an interrupt from direct memory access circuit 120 signaling the completion of the input data load operation. An interrupt service routine must be initiated to service the interrupt. In addition, such an interrupt would require a context switch from the interrupted process to the interrupt  
10 service routine, and another context switch to return from the interrupt. Consequently, the receive data synchronization command frees up considerable capacity within digital signal processor core for more productive use.

Another non-computational command is a send data synchronization command. The send data synchronization command is nearly the inverse of the receive data synchronization  
15 command, and actually asserts the signal specified. Upon reaching the send data synchronization command, reconfigurable IPP hardware co-processor 140 asserts a signal which then triggers a direct memory access operation. This direct memory access operation reads data from data memory 145 for storage at another system location. This direct memory access operation may be preset by digital signal processor core 110 and is merely begun upon  
20 receipt of a signal from reconfigurable IPP hardware co-processor 140 upon encountering the send data synchronization command. In the case in which direct memory access circuit 120 supports plural DMA channels, the send data synchronization command must specify the hardware signal that would trigger the correct DMA channel for the output data transfer. Alternatively, the send data synchronization command may specify the control parameters for  
25 direct memory access circuit 120, including the DMA channel if more than one channel is supported. Upon encountering such a send data synchronization command, reconfigurable IPP hardware co-processor 140 communicates directly with direct memory access circuit 120 to set up and start an appropriate direct memory access operation.

Another possible non-computational command is a synchronization completion  
30 command, actually another application of assert\_signal command. Upon encountering a synchronization completion command, reconfigurable IPP hardware co-processor 140 sends a signal to digital signal processor core 110. Upon receiving such a signal, digital signal processor core 110 is assured that all prior commands sent to reconfigurable IPP hardware co-processor 140 have completed. Depending upon the application, it may be better to sense this



5 signal via interrupt or by DSP core 110 polling a hardware status register. It may also be better to queue several operations for reconfigurable IPP hardware co-processor 140 using send and receive data synchronization commands and then interrupt digital signal processor core 110 at the end of the queue. This may be useful for higher level control functions by digital signal processor core 110 following the queued operations by reconfigurable IPP hardware co-processor 140. The IPP also uses the following other control/synchronization commands: Sleep; Reset; Write\_parameter. The write\_parameter command is used to perform parameter updates. Parameters that are changed frequently can be incorporated into commands to be specified on each task. Parameters, such as output right shift, additional term for rounding, saturation low/high bounds, saturation low/high set values, and operand size(8/16 bit), that are not often changed can be updated using the write\_parameter command.

The configurable IPP hardware co-processor supports the following computational commands directly:

- Row/column 8-point DCI/IDCI
- Vector addition/subtraction/multiplication
- 20 - Scalar-vector addition/subtraction/multiplication
- Table lookup
- Sum of absolute differences

In addition, through extension and special-casing of the above generic computational commands, the IPP also supports:

- 25 - 2 D DCI/IDCI
- demosaicing by simple interpolation
- chroma subsampling
- wavelets analysis and reconstruction
- color suppression
- 30 - color conversion
- memory-to-memory moves

Each command will include pointers for relevant data and coefficient storage(input data) as well as addresses for output result data. Additionally, the number of filter taps, up/down sampling factors, the number of outputs produced, and various pointer increment options are attached to the computational commands. Because image processing is the application area, 2-D block processing is allowed whenever feasible.

5        Figure 4 illustrates another possible arrangement of circuit 100. Circuit 100 illustrated in Figure 4 includes 2 reconfigurable IPP hardware co-processors, 140 and 180. Digital signal processor core operates with first reconfigurable IPP hardware co-processor 140 and second reconfigurable IPP hardware co-processor 180. A private bus 185 couples first reconfigurable IPP hardware co-processor 140 to reconfigurable IPP hardware co-processor 180. These co-  
10    processors have private memories sharing the memory space of digital signal processor core 110. The data can be transferred via private bus 185 by one co-processor writing to the address range encompassed by the other co-processor's memory. Alternatively, each co-processor may have an output port directed toward an input port of another co-processor with the links between co-processors encompassed in private bus 185. This construction may be  
15    particularly useful for products in which data flows from one type operation handled by one co-processor to another type of operation handled by the second co processor. This private bus frees digital signal processor 110 from having to handle the data handoff either directly or via direct memory access circuit 120.

      Alternatively, Figure 5 illustrates digital signal processor core 110 and a reconfigurable  
20    IPP hardware co-processor 140 loosely connected together via system bus 142. Digital signal processor core 110 may be of conventional design. In the preferred embodiment, reconfigurable IPP hardware co-processor 140 is adapted to coordinate with direct memory access circuit 120 for autonomous data transfers independent of digital signal processor core 110. The parts illustrated in Figure 5 are not intended to exclude the provision of other  
25    conventional parts. The system level connection in Figure 5 may be useful when the digital signal processor core 140 in a particular implementation does not offer connection to its internal bus, for example when using catalog devices. Data transfer overhead is usually larger when IPP coprocessor 140 is attached to the system bus, yet there is more system level flexibility, like using multiple DSPs or multiple IPPs in the same system, and relative ease of  
30    changing or upgrading DSP and IPP.

      As an example of the communication between and the DSP and the IPP, if the DSP is instructing the IPP to perform a vector addition task, these are the events that occur from the DSP's point of view. The DSP sets up the DMA transfer to send data to the IPP. Then the DSP sends a wait\_until\_ signal command to the IPP (this signal will be asserted by the DMA

- 5 controller once the transfer is completed). Next the DSP sends a vector\_add command to the IPP, which frees up the DSP to perform other tasks. Now, either the DSP comes back to check on the completion status of the IPP, or alternatively, the DSP can be interrupted upon completion of the IPP task upon receipt of a assert\_signal command, which would follow the vector\_add command. Finally, the DSP sets up the DMA to get the result back from the IPP.
- 10 As mentioned previously, as there is some overhead in managing each data transfer and each computation command, the functionality of the IPP supports and encourages block computations. Another advisable practice is to perform cascaded tasks on the IPP for the same batches of data, to reduce data transfers, and thus reduce the DSP load as well as the system bus load and overall power consumption.
- 15 The IPP supports one-dimensional, row-wise filtering when data is stored in rows. Certain combinations of upsampling and downsampling are supported as well. For example, the following 5 methods implement various up/down sampling options and constraints on filter length. Only configurations A and D (Figures 8 and 12) are considered here; there are many more methods in a fully reconfigurable IPP datapath (Figure 13).

20

Method	a) no up/down sampling	b) u/s up sample in space-time	c) up sample in space	d) - down sample in space	e) up sample in space-time
Configuration	A (8 MACs)	A (8 MACs)	A (8 MACs)	D (quad 2-trees)	D (quad 2-trees)
Filter taps (Util=1)	Any	any	any	Even	even
Upsampling factor	1	8, 16, 24	2, 4, 8	1	4, 8, 12
Downsampl factor	1	Any	1	2	Any

- Figures 6 - 15 illustrate the construction of an exemplary reconfigurable IPP hardware co-processor with Figures 8 and 10-15 illustrating various Datapath configurations. Figure 6 illustrates the overall block diagram general architecture of reconfigurable IPP hardware.
- 25 coprocessor 140 according to a preferred embodiment of the invention. On the host's memory map, the IPP interface should appear as large contiguous memory blocks, for coefficients, data and macro-commands, and also as discrete control/status registers, for configuration, command

5 queue, run-time control, etc. The configuration/command queue registers may very well sit on the host's DSP external bus in either I/O or memory address space. Multiple write addresses (with respect to the host) must be set up to modify less frequently changed parameters in IPP such as hardware handshake signaling, software reset, and so on. One write address for commands, links to an internal command queue. There are a few additional write addresses  
10 for clearing interrupts, one for each interrupt. There is at least one read address for query of command completion status.

The data portion should map into the host's memory space, if possible. If the address space is insufficient, address and data ports should be separate, such that writing to the address port sets up an initial address, and subsequent read/writes to the data port transfer contiguous  
15 data from/to the IPP data memory. In terms of IPP implementation, buffering is necessary between the outside 16/32 bit bus and the internal memory's 128 bit width. A small cache can be used for that purpose. Read ahead technique for reading and write-back for writing can reduce the access time. Around 512 bits in this buffer, half for read and half for write, should be sufficient.

20 Three logical memory blocks, data memory A and B and command memory, are accessible from a system bus via an external bus interface. The memory interface handles memory arbitration between the IPP 140 and the system bus 142, as well as simple First-In First-Out (FIFO) control involved in matching the system bus access width with the memory width. Data A and B are for input/output data and coefficients. Cascaded commands can  
25 reuse areas in the data memory, so the terms input/output are in the context of a single command. As previously mentioned, the Command Queue 141 can receive commands from the digital signal processor 110 via the digital signal processor bus 142, and in supplying those commands to the Execution Control unit 190, control the operation of the reconfigurable IPP hardware coprocessor 140. The control block steps through the desired memory access and  
30 computation functions indicated by the command. Command memory 141 is read by the decode unit 142. To conserve memory, variable length commands are incorporated. The decode unit 142 sends the produced control parameters (one set per command) to the execution control unit 190, which use the control parameters to drive a pipelines control path to fan out the control signals to the appropriate components. Control signals can be either fixed or time

5     varying in a command. They include memory access requests, input/output formatter control, and datapath control

       Data memory 145 and coefficient memory 147 are wide memory blocks (128-bit each) to support an 8-way parallel 16-bit datapath. This 128 bit wide memory block precludes the data path from having to access memory every cycle. The Data Memory 145 receives relevant  
10    input data via the DSP bus and also stores the Resultant Data subsequent processing through the Datapath core 170 and reformatting in the Output Formatter 180. Coefficient data can also be received from the DSP bus 142, or possibly, provided in a Look-Up Table within the IPP itself, and along with the input data, be processed through the Datapath core 170 and then reformatted in the Output formatter block 180. Data memory 145 and coefficient memory 147  
15    may be written to in 128 bit words. This write operation is controlled by digital signal processor core 110 or direct memory access circuit 120 which, through the use of operand pointers in the commands, manage the two memory blocks. Address generator 150 generates the addresses for recall of Data and Coefficients used by the co-processor. This read operation operates on data words of 128 bits from each memory.

20       The recalled 128 bit data words from Data and Coefficient Memories are supplied to input formatter 160. Input formatter 160 performs various shift and alignment operations generally to arrange the 128 bit input data words into the order needed for the desired computation. Input formatter outputs a 128 bit (8 by 16 bits) Data A, a 128 bit (8 by 16 bits) Data B and a 128 bit (8 by 16 bits) Coeff Data.

25       These three data streams, Data A, Data B, and Coeff Data, are supplied to Datapath 170. Datapath 170 is the operational portion of the co-processor. The datapath can be configured in the run-time to support a variety of image processing tasks. Figures 12 and 13 illustrate two preferred embodiments of the invention. Some tasks can be mapped into both configurations, each providing a different pattern of input/output memory access. These  
30    choices offer flexibility in the hand of application programmers to balance speed, data memory and sometimes power requirements. As will be further described below, datapath 170 includes plural hardware multipliers and adders that are connectable in various ways to perform a variety of multiply-accumulate operations. Datapath 170 outputs three adder data streams. Two of these three are 16 bit data words while one of the three is a 128 bit word(8 by 16 bits)

5           These three data streams supply the inputs to output formatter 180. Output formatter 180 rearranges the three data streams into eight 128 bit data words for writing back into the memory. The addresses for these two write operations are computed by address generator 150. This rearrangement may take care of alignment on memory word boundaries.

          The operations of co-processor are under control of control unit 190. Control unit 190  
10   recalls the commands from command queue 141 and provides the corresponding control within co-processor 140.

          The construction of input formatter 160 is illustrated in Figure 7. The two data streams Data A and Data B of 128 bits each are supplied to an input of multiplexers 205 and 207. Each multiplexer independently selects one input for storage in its corresponding register,  
15   215 and 217 respectively. Multiplexer 205 may select either one of the input data streams or to recycle the contents of register 215. Multiplexer 201 may select either the contents of register 215 or to recycle the contents of its register 211. Multiplexer 207 may select either the other of the input data streams, or to recycle the contents of register 217. The lower bits of shifter 221 are supplied from register 215. The upper bits of shifter 221 are supplied by  
20   register 211. Shifter 221 shifts and selects all 256 of its input bits and 128 bits are supplied to one full/4 way 64b x 2-1 multiplexer 231 and 128 bits are supplied to full/1way/4way 128b x 3-1 multiplexer 235. The 128 bit output of multiplexer 231 is stored temporarily in register 241 and forms the Data A input to datapath 170. The 128 bit output of multiplexer 235 is stored temporarily in register 245 and forms the Data B input to datapath 170. The output of  
25   multiplexer 207 is supplied directly to a full/1w/2w/4w 128b x 4-1 multiplexer 237 as well as supplied to register 217. Multiplexer 237 selects the entire 128 bits supplied from register 217 and stores the result in register 247. This result forms the coefficient data input to datapath 170.

          As mentioned previously, the three data streams, Data A, Data B, and Coeff Data, are  
30   supplied to Datapath 170 for processing. Figure 8 illustrates a Datapath architecture according to a first preferred embodiment of the invention, in which eight Multiply Accumulate Units (MACs) are connected in parallel ("A" configuration). The multiply-accumulate operation, where the sum of plural products is formed, is widely used in signal processing, for example, in many filter algorithms. N multiply accumulate (where N=8 in this example) units are

5 operated in parallel to compute  $N$  output points. This configuration is suitable for a wide-memory word that contains multiple pixels, typical for image processing. The feedback loop on the final row of adders contain multiple banks of accumulators to support upsampling. According to a preferred embodiment, each MAC is associated with 3 accumulators, and Control Unit 190 includes the necessary addressing mechanism for these accumulators. An  
 10 accumulator depth of three is chosen in order to support color conversion, which involves  $3 \times 3$  matrixing. Thus, an accumulator depth of three simplifies implementation for color conversion.

Figure 9 illustrates the construction of the output formatter 180 illustrated in Figure 6. The 16 bit dataword outputs of the first and second accumulators within reconfigurable IPP  
 15 hardware co-processor 140 (Acc[0] and Acc[1]) form the first two inputs to the output formatter 180, with the outputs of all 8 accumulators of reconfigurable IPP hardware co-processor 140 (Acc[0], Acc[1], Acc[2], Acc[3], Acc[4], Acc[5], Acc[6], Acc[7]) providing the third input to the output formatter. Eight, 16 bit blocks are written to data memory 145 subsequent processing through the multiplexers and registers of output formatter 180.

20 Figure 10 illustrates the construction of datapath 170 according to a second preferred embodiment illustrating a single 8-tree adder configuration ("B" configuration). Various segments of the Data A and Data B 128 bit ( $8 \times 16$  bit) dataword inputs to the datapath 170, supplied from input formatter 160, are supplied to adders/subtractors (adders), 310, 320, 330, 340, 350, 360, 370 and 380. As shown, the first 16 bit datawords, Data A[0] and Data B[0],  
 25 which represent the left most or most significant bits of the 128 bit output, are coupled to adder 310, and adder 320, the second 16 bit datawords Data A[1] and Data B[1] are coupled to adder 330 and adder 340, the third 16 bit datawords, Data A[2] and Data B[2] are coupled to adder 350 and adder 360, the fourth 16 bit datawords, Data A[3] and Data B[3] are coupled to adder 370 and adder 380. The result of this addition or subtraction of the first 16 bit datawords  
 30 through fourth datawords is stored in pipeline registers 312, 322, 332, 342, 352, 362, 372 and 382. This result is then multiplied by the Coeff Data, which for this configuration of IPP, consists of the same two 16 bit datawords. In other words, with the 8 MAC configuration shown in Figure 10, 4 data words and two coefficient words are fed to the hardware, on each cycle. These same two coefficient words are used in every pair of adders to multiply the input

5 data point with, and the products, which are stored in pipeline registers 316, 326, 336, 346, 356, 366, 376 and 386, are summed in adders 318, 338, 358 and 378. The results of those summations are summed in adders 328 and 368, the summations of which are added in adder 348. The output of adder 348 is accumulated in accumulator 349. The benefit of this configuration is the requirement of only, albeit 8 multipliers, one accumulator to process the  
10 two 128 bit word outputs of input formatter 160.

Figure 11 illustrates the construction of datapath 170 according to a third preferred embodiment illustrating a dual 4-tree with butterfly adder configuration ("C configuration"). Various segments of the Data A and Data B 128 bit ( $8 \times 16$  bit) dataword inputs to the datapath 170, supplied from input formatter 160, are supplied to adders/subtractors (adders), 310, 320, 15 330, 340, 350, 360, 370 and 380. As shown, the first 16 bit datawords, Data A[0] and Data B[0], which represent the left most or most significant bits of the 128 bit output, are coupled to adder 310, the second 16 bit datawords Data A[1] and Data B[1] are coupled to adder 320, the third 16 bit datawords, Data A[2] and Data B[2] are coupled to adder 330, the fourth 16 bit datawords, Data A[3] and Data B[3] are coupled to adder 340, the fifth 16 bit datawords, Data  
20 A[4] and Data B[4] are coupled to adder 350, the sixth 16 bit datawords Data A[5] and Data B[5] are coupled to adder 360, the seventh 16 bit datawords Data A[6] and Data B[6] are coupled to adder 370 and the eighth 16 bit datawords, or the least significant bits of the 128 bit output of input formatter 160, Data A[7] and Data B[7] are coupled to adder 380. The result of this addition or subtraction of first 16 bit datawords through eighth datawords is stored in  
25 pipeline registers 312, 322, 332, 342, 352, 362, 372 and 382. This result is then multiplied by the Coeff Data, which for this configuration of IPP, consists of two 16 bit words. In other words, with the 2 MAC configuration shown in Figures 11, 8 datawords and two coefficient words are fed to the hardware, on each cycle. These same two coefficient words are used in every adder/multiplier portion of each MAC unit to multiply the input data point with, and the  
30 products, which are stored in pipeline registers 316, 326, 336, 346, 356, 366, 376 and 386, are summed in adders 318, 338, 358 and 378. The results of those summations are summed in adders 328 and 368. The summation from adder 328 is then subtracted from the summation from adder 368 in subtractor 388. The output from 388 is then accumulated in accumulator 359. The summation from adder 368 is then added to the summation from adder 328 in adder



5 348. The output of adder 348 is then accumulated in accumulator 349. The output of adder 348 is accumulated in accumulator 349. The benefit of this configuration is the requirement of only, albeit 8 multipliers, two accumulators to process the two 128 bit word outputs of input formatter 160.

10 Figure 12 illustrates the construction of datapath 170 according to a fourth preferred embodiment wherein a quad 2-tree adder configuration is illustrated ("D configuration"). Various segments of the Data A and Data B 128 bit(8 x 16 bit) dataword inputs to the datapath 170, supplied from input formatter 160, are supplied to adders/subtractors (adders), 310, 320, 330, 340, 350, 360, 370 and 380. Two different input data schemes are envisioned. The first scheme provides 8 datawords and 2 coefficient words to the hardware each cycle.

15 Downsampling of 2x is performed with the filtering. Each pair of MAC units performs two multiplications and accumulates the sum of the products. The second scheme provides 2 datawords and 8 coefficient words to the hardware each cycle. Again, each pair of MAC units performs two multiplications, an addition and an accumulation. Upsampling is performed with the 4-way parallelism and optionally with the depth of each accumulator.

20 According to the first scheme, the first 16 bit datawords, Data A[0] and Data B[0], which represent the left most or most significant bits of the 128 bit output, are coupled to adder 310, the second 16 bit datawords Data A[1] and Data B[1] are coupled to adder 320, the third 16 bit datawords, Data A[2] and Data B[2] are coupled to adder 330, the fourth 16 bit datawords, Data A[3] and Data B[3] are coupled to adder 340, the fifth 16 bit datawords, Data A[4] and Data B[4] are coupled to adder 350, the sixth 16 bit datawords Data A[5] and Data B[5] are coupled to adder 360, the seventh 16 bit datawords Data A[6] and Data B[6] are coupled to adder 370 and the eighth 16 bit datawords Data A[7] and Data B[7] are coupled to adder 380. The result of this addition or subtraction of first bit datawords through eighth datawords is stored in pipeline registers 312, 322, 332, 342, 352, 362, 372 and 382. This

25 result is then multiplied by the Coeff Data, which for this configuration of IPP, consists of two 16 bit coefficient words. In other words, with the quad 2-tree adder configuration shown in Figure 12, 8 datawords and two coefficient words are fed to the hardware, on each cycle. The same two coefficient words are used in every pair of MAC units to multiply the input data point with, and the products, which are stored in pipeline registers 316, 326, 336, 346, 356,

30

5 366, 376 and 386, are summed in adders 318, 338, 358 and 378. The summation from adders 318, 338, 358 and 378 are then accumulated in accumulators 319, 339, 359 and 379. The benefit of this configuration is the requirement of only, albeit 8 multipliers, four accumulators to process the two 128 bit word outputs of input formatter 160.

Figure 13 illustrates the construction of datapath 170 that includes routing and  
 10 multiplexing necessary to support the 4 configurations, A, B, C, and D (Figures 8, 10, 11, and 12). Various segments of the Data A and Data B 128 bit(8 x 16 bit) dataword inputs to the datapath 170, supplied from input formatter 160, are supplied to adders/subtractors (adders), 310, 320, 330, 340, 350, 360, 370 and 380. As shown, the first 16 bit datawords, Data A[0] and Data B[0], which represent the left most or most significant bits of the 128 bit output, are  
 15 coupled to adder 310, the second 16 bit datawords Data A[1] and Data B[1] are coupled to adder 320, the third 16 bit datawords, Data A[2] and Data B[2] are coupled to adder 330, the fourth 16 bit datawords, Data A[3] and Data B[3] are coupled to adder 340, the fifth 16 bit datawords, Data A[4] and Data B[4] are coupled to adder 350, the sixth 16 bit datawords Data A[5] and Data B[5] are coupled to adder 360, the seventh 16 bit datawords Data A[6] and Data  
 20 B[6] are coupled to adder 370 and the eighth 16 bit datawords Data A[7] and Data B[7] are coupled to adder 380. The result of this addition or subtraction of first bit datawords through eighth datawords is stored in pipeline registers 312, 322, 332, 342, 352, 362, 372 and 382. This result is then multiplied by the Coeff Data, which for this configuration of IPP, consists of the same 16 bit dataword. In other words, with the 8 MAC configuration shown in Figures  
 25 8 and 13, 8 datawords and one coefficient dataword is fed to the hardware, on each cycle. This same coefficient dataword is used in every MAC unit to multiply the input data point with, and the products, which are stored in pipeline registers 316, 326, 336, 346, 356, 366, 376 and 386, are accumulated in adders 318, 328, 338, 348, 358, 368, 378 and 388.

Actually, as shown in the routing and multiplexing for configurations A/B/C/D diagram  
 30 of Figure 13, the products form one input to adders, 318 through 388. The second input to adder 318 is formed by the output of multiplexer 319, which has two inputs; the first being the product from the multiplier 324 and the second being the accumulated sum of adder 318. Adder 328 has multiplexers 325 and 329 on both inputs. Multiplexer 325 selects between multiplier 324 or the output of adder 318. Multiplexer 329 selects between accumulated result

5 from adder 328 itself, or from the next adder 338. In the 8 MACs configuration (A, Figure 8), the pair of adders 318 and 328 implement separate accumulation of products from multipliers 314 and 324. In the quad 2-trees configuration (E, Figure 12), the pair of adders 318 and 328 implement summation of the products (by 318) then accumulating the sums (by 328)

10 Similarly, the adder pair 338 and 348, the adder pair 358 and 368, and the adder pair 378 and 388 each implement either separate accumulation of products or accumulation of sums of 2 products. In case of the summed up accumulation supporting quad 2-trees configuration, adders 348, 368, and 388 produces the final accumulated outputs, just like adder 328

To support the dual 4-tree with butterfly configuration (C), multiplexers 319, 339, 359, 15 and 379 are selected such that adders 318, 338, 358, and 378 sums up neighboring pairs of products from the 8 multipliers. Multiplexers 325 and 329 are selected such that adder 328 adds up results of adders 318 and 338, and thus has the sum from the first 4 multipliers 314, 324, 334, and 344. Multiplexers 365 and 369 are similarly selected so that adder 368 has the sum from the last 4 multipliers 354, 364, 374 and 384. These 2 sums, at adders 328 and 368, 20 are then routed to both adders 348 and 390, which implement the cross add/subtract operations. Adder 348 performs the addition, and adder 390 performs the subtraction. Results from adders 348 and 390 are next routed to adders 388 and 392, respectively, for accumulation. Adders 388 and 392 produces the final pair of outputs

To support the single 8-tree configuration (B), all multiplexer configuration for dual 4- 25 tree with butterfly configuration (C) is retained. Adder 348 has the sum from all 8 multipliers, and adder 388 has the accumulated result. Output of adder 392 is simply ignored.

Figure 14 illustrates a simplified version of reconfigurable datapath architecture. This simplified architecture supports both the parallel MACs of Figure 8 and the quad 2-trees of Figure 12. As is shown, instead of the separate adders and multipliers illustrated in figures 8 and 13, both Data A and Data B inputs are applied to both a multiplier and an adder/subtractor (adder) and then the outputs of either the adders or multipliers are selected before going out of 30 the multiply/add/subtract blocks 810, 820, 830, 840, 850, 860, 870, 880. A more in depth illustration of a pair of the MAC units of Figure 14 is shown in Figure 28. Each MAC unit is capable of performing a pipelined single cycle multiply accumulate operation on two inputs

5 D\_inp and C\_inp Accumulation of  $D\_inp + C\_inp$  or  $D\_inp - C\_inp$  instead of  $D\_inp * C\_inp$  is also possible, hence the add/subtract unit 310 placed in parallel with each multiplier 314. The multiplexer 610 chooses between the adder/subtractor 310 output or the multiplier 314 output. Between each pair of MAC units, there is also the quad 2-trees option (indicated by the AND gate 710) to add up the pair of results ( $D\_inp */+/- C\_inp$ ), to produce ACC\_inp, 10 which feeds the accumulating adder 818.

As shown in Figure 14, both of the above described configurations are implemented. Although only 8 adders (excluding those in parallel with multipliers) are active at any given time, 12 physical adders are used in this design, in order to reduce the cost of multiplexing and routing. The AND gates 710, 720, 730 and 740 on the cross path control whether or not the 15  $*/+/-$  results should be added together. As shown in Figure 28, three accumulators 612, 614 and 616 are available in each MAC unit to implement upsampling. The accumulator 818 can select, via multiplexer 618, any of the three as input (with the other input being ACC\_inp), or from the half-unit quantity for rounding, RND\_ADD. On the very first cycle of valid data on ACC\_inp, RND\_ADD should be the selected input.

20 Rounding and saturation follow the main arithmetic datapath. With the half-unit quantity already added to the accumulated sum, rounding is simply a right shift. Figure 15 illustrates a more simplified version of Figure 8 than that illustrated in Figure 14. The configuration illustrated in Figure 15 comprises only 4 MAC units versus 8 MAC units illustrated in previous configurations and does not contain the pre-add illustrated in Figures 8- 25 14. As illustrated in Figures 14 and 28, Figure 15 illustrates Data A and Data B inputs applied to both a multiplier 314 and an adder/subtractor (adder) 310 and then the outputs of the adders and multipliers are multiplexed together in multiplexers 610 and 620 (Figure 28). Because there is no pre-add, post multiplexing, the outputs of the multiplexers 610 and 620 are accumulated in accumulators, 818, 828, 838 and 848. As previously described with reference 30 to Figure 14, and as shown in Figure 28, three accumulators 612, 614 and 616 are available in each MAC unit to implement upsampling. The accumulator 818 can select, via multiplexer 618, any of the three as input (with the other input being ACC\_inp), or from the half-unit quantity for rounding, RND\_ADD. On the very first cycle of valid data on ACC\_inp, RND\_ADD should be the selected input.

5 In Figures 14 and 15, it is sometimes desirable to add absolute difference operation to the multiply/add/subtract block. This will speed up motion estimation task in video encoding applications.

Figure 16 illustrates the input data formatting necessary to perform the IPP operation of row filtering. On the first cycle, the Data A input to all 8 MACs comprises the first 8 data words. Every cycle, the window of input data words used to feed the MACs is shifted one word to the right. Data B input of all 8 MACs is fed the same coefficient word. In this example, a 3-tap FIR filter is implemented, so three coefficient words are provided.

10 In the figure,  $X_0 \dots X_7$  comprise the first Data A input to the MACs during a first clock cycle. Shifting by one data word, the second Data A input becomes  $X_1 \dots X_8$  during a second clock cycle. The Data A inputs continue in this manner, supplying each MAC with consecutive sequence of data words. The first filter coefficient  $C_0$  is broadcast to all MACs for the first cycle.  $C_1$  is broadcast to all MACs for the second cycle, and  $C_2$  for the third cycle. At the third cycle, the MAC units have accumulated the correct outputs and can write back results to data memory. The data feed continues at  $X_8 \dots X_{15}$  to begin to compute output.

20  $Y_8 \dots Y_{15}$ , and the coefficient feed wraps back to  $C_0$ .

Maintaining the same configuration, an alternative output is rendered when instead of supplying 8 data words and one coefficient word to the hardware, providing one data word and 8 coefficients words for the 8 filter banks. Again each Mac is working independently, multiplying the same data word with its specific coefficient word and accumulating the products. Upsampling is performed with the 8-way parallelism and optionally with the depth of each accumulator. Figure 17 illustrates the input data formatting necessary to perform a symmetric row filtering operation. In this example IPP implements a 3-tap filter, so the first and third coefficients are equivalent. Therefore, only two coefficient words are provided. On the first cycle, the Data A input comprises the first 8 data words  $X_0 \dots X_7$ . The first Data B input comprises data words  $X_2 \dots X_9$ . In addition, the first coefficient supplied to all the multipliers is  $C_0$ . The second Data A input is the first Data A input shifted to the right one word, or  $X_1 \dots X_8$ . The second Data B input is the same 8 data words. Coefficient  $C_1$  is supplied to all the multipliers on the second cycle. Effectively, IPP computes

30

$$C_0 \cdot (X_0 + X_2) + 2 \cdot C_1 \cdot X_1 \text{ on the first MAC,}$$

5  $C_0(X_1 + X_3) + 2 \cdot C_1 \cdot X_2$  on the second MAC,

and so on. Let the desired filter coefficients be  $F_0, F_1, F_2$ , where  $F_0 = F_2$ . The supplied coefficients should relate to the desired coefficients by

$$C_0 = F_0$$

$$C_1 = 0.5 \cdot F_1$$

10 At the end of the second cycle, the 3-tap filter outputs are ready to be stored back to data memory. On the third cycle, the Data A input is supplied with data words  $X_3 \dots X_{15}$ , Data B input is supplied with  $X_{16} \dots X_{17}$ , and coefficient is wrapped back to  $C_0$ .

Figure 18 illustrates where from in memory the data comes to perform a column filter operation. The computational model and command syntax is similar to the row filter computational model and command syntax, except that data is stored in row-major order, and inner products are performed along columns. For best efficiency, data, coefficient and output arrays should all be aligned to a 8 x 16 bit memory word. As is shown in Figure 18, in this case the already aligned data is taken directly from memory word to the datapath. In other words, no input formatting of the data is necessary. Each coefficient is applied to all 8 MAC units in the parallel MACs configuration shown in Figures 8 and 10 through 13. An N tap column filter takes N+1 cycles to produce 8 outputs. There are N memory reads and 1 data memory writes in each N+1 cycles. When  $N > 8$ , there is one coefficient memory read every 8 cycles. Otherwise there is an initial read then all subsequent coefficients are supplied by the register in input formatter; no further read is needed. Coefficient read frequency is the same as in row filtering, 1 read/8 cycles if  $N > 8$ , and is zero otherwise.

Figure 19 illustrates the IPP configuration necessary to perform the sum of absolute differences used to enhance the performance of video encoding. As shown in Figure 19, Data A comprises  $X_0 \dots X_7$  and Data B comprises  $Y_0 \dots Y_7$ . Coefficient words are not required. The difference between each Data A input and each Data B input is calculated in subtractors 310, 320, 330, 340, 350, 360, 370 and 380 and those differences are stored in registers 312, 322, 332, 342, 352, 362, 372 and 382. That difference is then multiplied by either a plus or a minus sign depending upon whether the difference is positive or negative in multipliers 314, 324, 334, 344, 354, 364, 374 and 384, in order to yield a positive number. Those products are stored in registers 316, 326, 336, 346, 356, 366, 376 and 386 then summed in adders 318,

5 328, 358 and 378 and those sums summed in adders 328, 348 and 368. The sum of adder 348 is then accumulated in accumulator 349. For the sum of absolute differences we operate on 8-bit pixels, so the adders only have to be 12-bits wide, except for the final accumulator, which must be 16 bits wide. Saturation thresholds and rounding parameters can come from yet another bank of registers.

10 Figures 20, 21 and 22 illustrate the IPP operation of Discrete Sine/Cosine Demosaicing including the steps of Row Pass and Column Pass. Most digital still cameras employ color filter array in the imager that produces interleaved color information. Demosaicing is the process to obtain the missing color component from available neighboring same-color components. Simple linear interpolation approach is often used, which can be represented by  
15 the diagram illustrated in Figure 20. The weights are either 0.5 or 0.25, depending upon whether there are 2 or 4 closest same-color neighbors (excluding boundary conditions).

The three colors are processed separately, with red processing essentially the same as blue. Each color is processed in two passes, a row pass and a horizontal pass. The row pass is graphically represented in Figure 21. From each green/red line, one full green line and one  
20 full red line is generated. For the green component, row pass filtering is implemented by a 2-phase, 3-tap filter, with coefficients (0.5, 0, 0.5) and (0, 1, 0) for the two phases. For the red component, row pass filtering is implemented by the same 2-phase, 3-tap filter, with coefficients (0, 1, 0) and (0.5, 0, 0.5). Each blue/green line is processed similarly to generate a full blue line and a full green line.

25 Producing two color output rows from one row should be merged into one command, using up-sampling-like looping. It takes 6 cycles to process 8 input pixels. For each group of 6 cycles, there is one data memory read, two data memory writes, and three coefficient memory reads.

The implementation of column pass for demosaic red/blue components is illustrated in  
30 Figure 22a. For red and blue colors, two tap column filtering is used. It takes three cycles to process 8 input pixels during which there are two data memory reads, 1 data memory writes, and there are no steady-state coefficient memory reads.

The implementation of column pass for demosaic green components is illustrated in Figure 22b. For the green color component, 2-phase 3-tap column filtering is used, with

5 coefficients (0.25, 0.5, 0.25) and (0, 1, 0). Eight input pixels are processed in 4 cycles. There are three data memory reads, one data memory write, and zero coefficient memory reads per group of 4 cycles.

In sum, 11 cycles are spent for the interpolation scheme of demosaic for 8 input pixels. Out of 13 cycles, 6 data memory reads, 4 data memory writes and 3 coefficient memory reads are performed.

Figure 23 illustrates the formatting of the input data to perform the IPP operation of wavelets, row pass. In image technology, wavelets are used for image compression/decompression and feature extraction, for example, as a pre-processing stage for textural features. The wavelets operation can be implemented on any of the parallel 8 MAC configurations illustrated in Figures 8 and 10-13 or the more simplified versions of Figures 14 and 15. The row pass of wavelets analysis is implemented as 2x upsampling, 2x downsampling (to achieve high/low frequency banks), row filtering.

Figure 24 illustrates where from, in memory, the input data comes, in order to perform the column pass portion of the wavelet operation. The column pass is treated as 2x upsampling, 2x downsampling, column filtering. Again, data, coefficient and output arrays should all be aligned to a 8 x 16 bit memory word. As is shown in Figure 18, data is taken directly from memory word to the datapath. In other words, no input formatting of the data is necessary. Each coefficient is applied to all 8 MAC units in the parallel MACs configuration shown in Figures 8 and 10 through 13 or to the four MAC units illustrated in figures 14 and 15. It takes  $N+1$  cycles to produce 8 outputs, where  $N$  is the number of filter taps in the wavelets kernel. There are  $N$  memory reads and 1 data memory writes in each  $N+1$  cycles. Coefficient read frequency is the same as in row filtering, 1 read/8 cycles if  $N > 8$ , and is zero otherwise. For wavelet reconstruction, separately process high and low frequency banks with 2x upsampling filters. Finally, combine the two banks using vector addition.

Figure 25 illustrates the IPP operation of Indirect Cosine Transform (IDCT) in a row pass format. As shown, row-pass IDCT is implemented with the full matrix-vector approach. Thirty-two multiplications are used for each 8-point transform. Although not seemingly very efficient, a straightforward application of the IPP. Any one of the 8 MAC configurations shown in Figures 8 or 10-15 can be used to perform this operation, but the configuration of the



5 split adder trees with butterfly shown in Figure 11 is preferred. This configuration can take advantage of symmetry in the transform to reduce the number of multiplications by half. In this case the IPP uses the post-multiply/adders to implement the cross additions/subtractions. One input dataword is pulled from the wide memory word per cycle, and 8 coefficient words are used per cycle. Each 8-point transform takes 4 cycles to process. During these 4 cycles, 10 one data memory read, one data memory write and 4 coefficient memory reads are performed. If the butterfly stage of reconfiguration is omitted (for example in Figures 14 and 15), the full 8-by-8 matrix multiplication method has to be used, resulting in 64 multiplications per 8 point transform, and taking 8 or 16 cycles to perform each transform (with 8 or 4 MACs in IPP). Figure 26 illustrates the IPP operation of Direct Cosine Transform (DCT) in a row pass format. Similar to the row-pass IDCT, row-pass DCT can be implemented with 32 15 multiplications or with 64 multiplications, depending on the configurability of IPP. When the dual 4-tree with pre-multiply adders configuration (Figure 11) is available, it should be used. The butterfly stage is disabled in this case. All 8 data words from each memory word are applied to the MACs, one to each. Coefficients are applied the same way, one different 20 coefficient to each MAC. It takes 4 cycles to process one 8-point transform in this configuration. Without the pre-multiply adders (for example in Figures 14 and 15), each 8-point transform will require 64 multiplications, and take 8 or 16 cycles depending on the number of MACs in the IPP.

Figure 27 illustrates the IPP operation of IDCI in column format Single Instruction 25 Multiple Data (SIMD). The parallel configuration of 8 MACs shown in Figures 8 with some modifications in the accumulators is needed to take advantage of symmetry in the transform. Each MAC unit requires 8 accumulators, and each accumulating adder needs to take both inputs from the 8 accumulators. With such hardware capability, during the first 4 cycles, one 4 x 4 matrix will yield the first 4 points. During the next 4 cycles, another 4 x 4 matrix will 30 produce the next 4 points. During cycles 9 and 10, the accumulating adders cross add/subtract and combine the outputs. Therefore, in 10 cycles, a pair of output results, 16 points are produced. During those 10 cycles, 8 data reads, 2 data writes and 8 coefficient reads are performed. Without the hardware modification, it takes 64 multiplications per 8-point

5 transform, so 16 points of output will take 16 cycles on 8-MAC version of IPP, and 32 cycles on 4-MAC version of IPP. In either case the separate MAC configuration is used.

In addition to the datapath configurability and input formatting options, an efficient control and address generation scheme is devised for IPP. This scheme reduces the implementation cost of hardware control, and provides easy-to-use programming model for

10 IPP

All computation shall occur inside a nested for loop. Timing for accumulator initialization and write out shall be controlled by conditioning on the loop variables. Initialization shall happen when certain loop variables match with their beginning values. Write out shall happen when the same set of variables match with their ending values.

15 Circulating accumulators can be specified with the innermost loop count indexing the accumulators. All address increments for input data, coefficients, and results, can be specified in terms of "when" and "how much", and the "when" is associated with the loop variables. The following is pseudo-code of a skeleton of control structure for IPP that illustrates these concepts.

20

```

dptr = dptr_init; /* initial value of pointers */
cptr = cptr_init;
optr = optr_init;

```

25

```

for (i1=0; i1<=lp1end; i1++) {
  for (i2=0; i2<=lp2end; i2++) {
    for (i3=0; i3<=lp3end; i3++) {
      for (i4=0; i4<=lp4end; i4++) {

```

30

```

        /* memory read and input formatting */
        x[0..7] = dptr[i1..i7];
        /* or dptr[0], dptr[0,1], dptr[0,1,2,3] distributed */
        y[0..7] = cptr[i1..i7];
        /* or cptr[0], cptr[0,1], etc */

```

35

```

        /* accumulator initialization */
        if (initialize_acc)
            acc[i4*accmode][0..7] = rnd_add[0..7];

```

40

```

        /* operation-accumulate */
        acc[i4*accmode][0..7] += x[0..7] op y[0..7];

        /* write back */

```

```

5      if (write_back)
          optr[0..7] = saturate_round(acc[i4*accmode][0..7]);
          /* or just 1, 2, or 4 outputs */

          /* pointer updates */
10     dptr += ...;
       cptr += ...;
       optr += ...;

15     }
  }
}

```

The initialize\_acc condition is tested by matching a specified subset of loop count variables with the beginning values (0). The parameter acc\_loop\_level indicates whether none, i4, i4 and i3, or i4, i3 and i2 should be tested. This same subset of loop count variables are tested against their ending values to supply the write\_back condition.

The pointer updates also involve comparing loop count variables. For example, for 4 level of loops we can supply up to 4 sets of address modifiers for the data pointer, *dptr*. Each set consists of a subset of loop count variables that must match with their ending value, and the amount in which *dptr* should be incremented when the condition is true. The same capability is given to coefficient pointer *cptr* and output pointer *optr*.

In the above pseudo code, the parameters are used which are either statically set with Write\_parameters command or are encoded in an IPP computational command. These parameters includes the ending values of loop count variables (beginning value is always 0), accmode (single/circulating accumulators), op (multiply/add/subtract/absdiff), acc\_loop\_level and the address modifiers mentioned above.

All the supported imaging/video functions can be written in the above form and then translated into IPP commands by properly setting the parameters. The task of software development for IPP can follow this methodology.

5 I claim:

1. An image processing peripheral comprising:

a plurality of pairs of multiply accumulate circuits connected in parallel, each pair of multiply accumulate circuits comprising;

10 first adder pairs, each one of each adder pair having first and second inputs receiving respective first and second inputs having a first predetermined number of bits and an output producing a sum or a difference of said inputs;

first multiplier pairs, corresponding to said first adder pairs, each multiplier of each multiplier pair having a first input of said sum or difference of said first adders and a second input of a constant predetermined number and  
15 producing a product output;

second adder pairs, corresponding to said first multiplier pairs, each one adder of said adder pair having first and second inputs receiving respective first multiplier outputs from one or the other of said multipliers of said  
20 corresponding multiplier pair as said first input and

wherein said one of said pair of second adders receives an output from a multiplexer said multiplexer having one input from a product of the other multiplier of said first multiplier pairs and a second input from an accumulated sum of said one adder of said second adder pairs as a second input of said one  
25 adder of said second adder pair and;

wherein said other of said pair of second adders receives outputs from a first and a second multiplexer, said first multiplexer having one input from said other multiplier of said first multiplier pair and a second input from the sum of said one adder of said second adder pair, said second multiplexer having one  
30 input from the accumulated sum of said other adder of said second adder pair and a second input from the sum of a one adder of a second pair of second adder pairs, as a second output, and;

wherein said second adder pairs produce a sum output

- 5           2   An image processing peripheral
- a plurality of pairs of multiply accumulate circuits connected in parallel, each pair of multiply accumulate circuits comprising;
- first adder pairs, each one adder of each adder pair having first and second inputs receiving respective first and second inputs having a first
- 10           predetermined number of bits and an output producing a sum or a difference of said inputs;
- first multiplier pairs, corresponding to said first adder pairs, each multiplier of each multiplier pair having a first input of said sum or difference of said first adders and a second input of a constant predetermined number and
- 15           producing a product output;
- second adder pairs, corresponding to said first multiplier pairs, each adder pair implementing separate accumulation of said products of said first multiplier pairs, yielding an accumulated sum
- 20           3   An image processing peripheral comprising:
- a plurality of pairs of multiply accumulate circuits connected in parallel, each pair of multiply accumulate circuits comprising;
- first adder pairs, each one adder of each adder pair having first and second inputs receiving respective first and second inputs having a first
- 25           predetermined number of bits and an output producing a sum or a difference of said inputs;
- first multiplier pairs, corresponding to said first adder pairs, each multiplier of each multiplier pair having a first input of said sum or difference of said first adders and a second input of a constant predetermined number and
- 30           producing a product output;
- second adder pairs, corresponding to said first multiplier pairs, each adder pair implementing summation of said products of said pairs of multipliers and then accumulating the sums of said summations

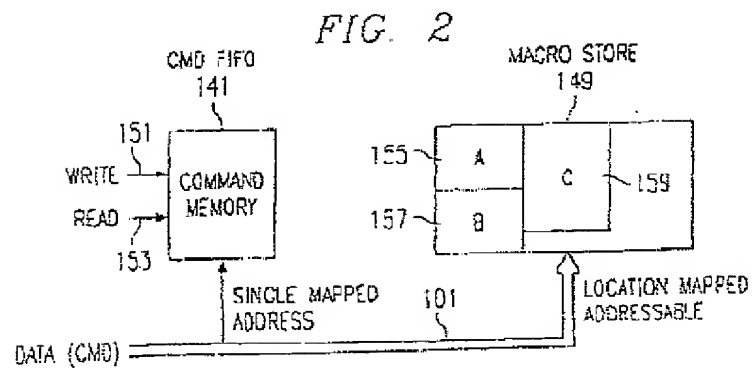
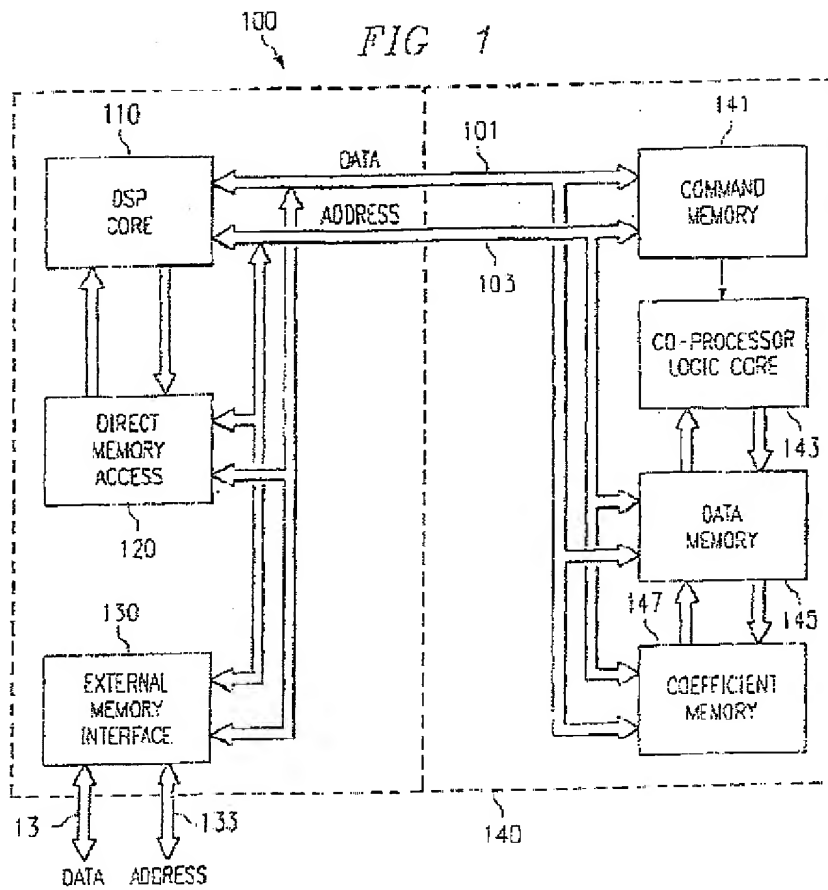
5        4. An image processing peripheral comprising:

         a plurality of pairs of multiply accumulate circuits connected in parallel,  
         each pair of multiply accumulate circuits comprising;

         first adder pairs, each one adder of each adder pair having first and  
         second inputs receiving respective first and second inputs having a first  
10        predetermined number of bits and an output producing a sum or a difference of  
         said inputs;

         first multiplier pairs, corresponding to said first adder pairs, each  
         multiplier of each multiplier pair having a first input of said sum or difference  
         of said first adders and a second input of a constant predetermined number and  
15        producing a product output;

         second adder pairs, corresponding to said first multiplier pairs, each  
         adder pair implementing accumulation of sums of two products



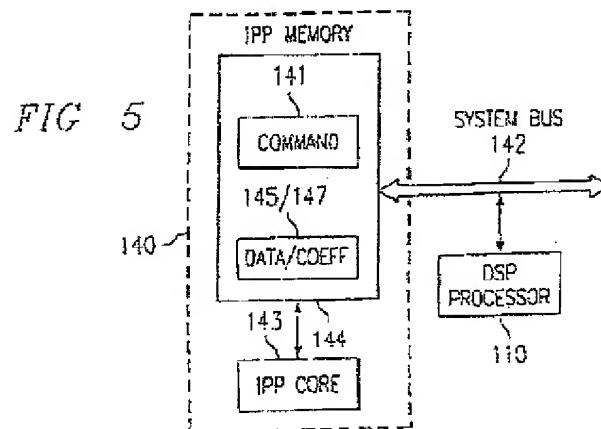
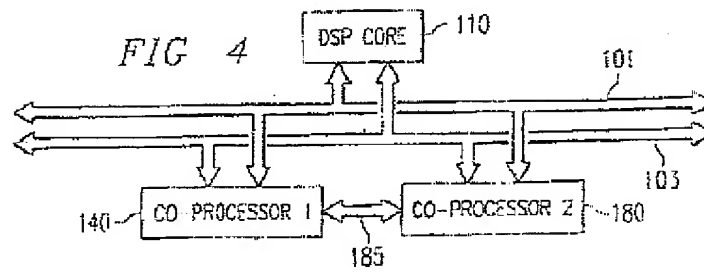
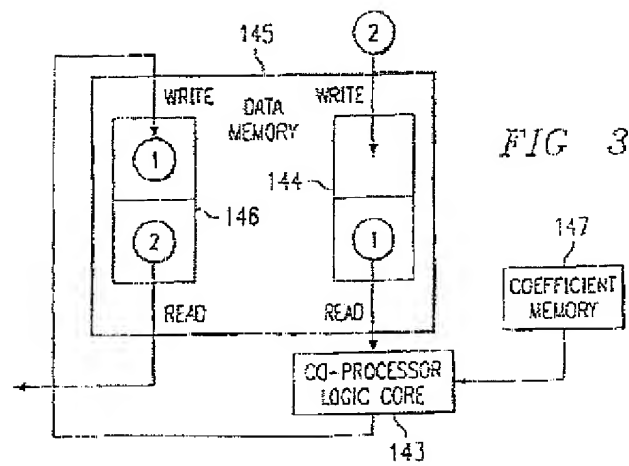
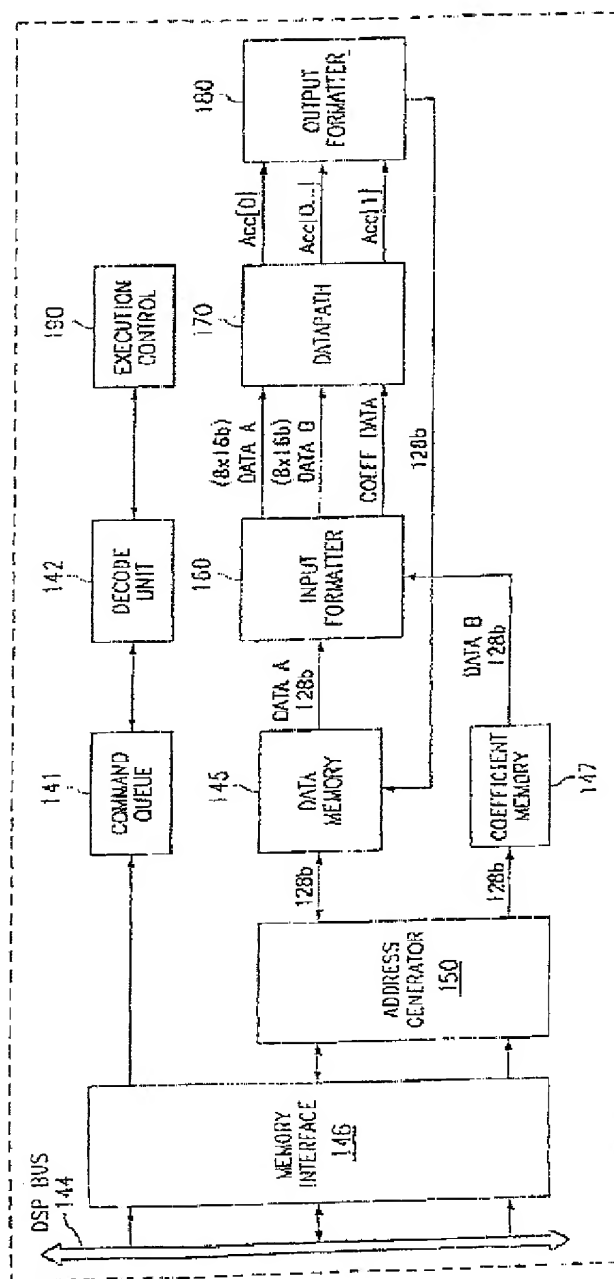




FIG. 6



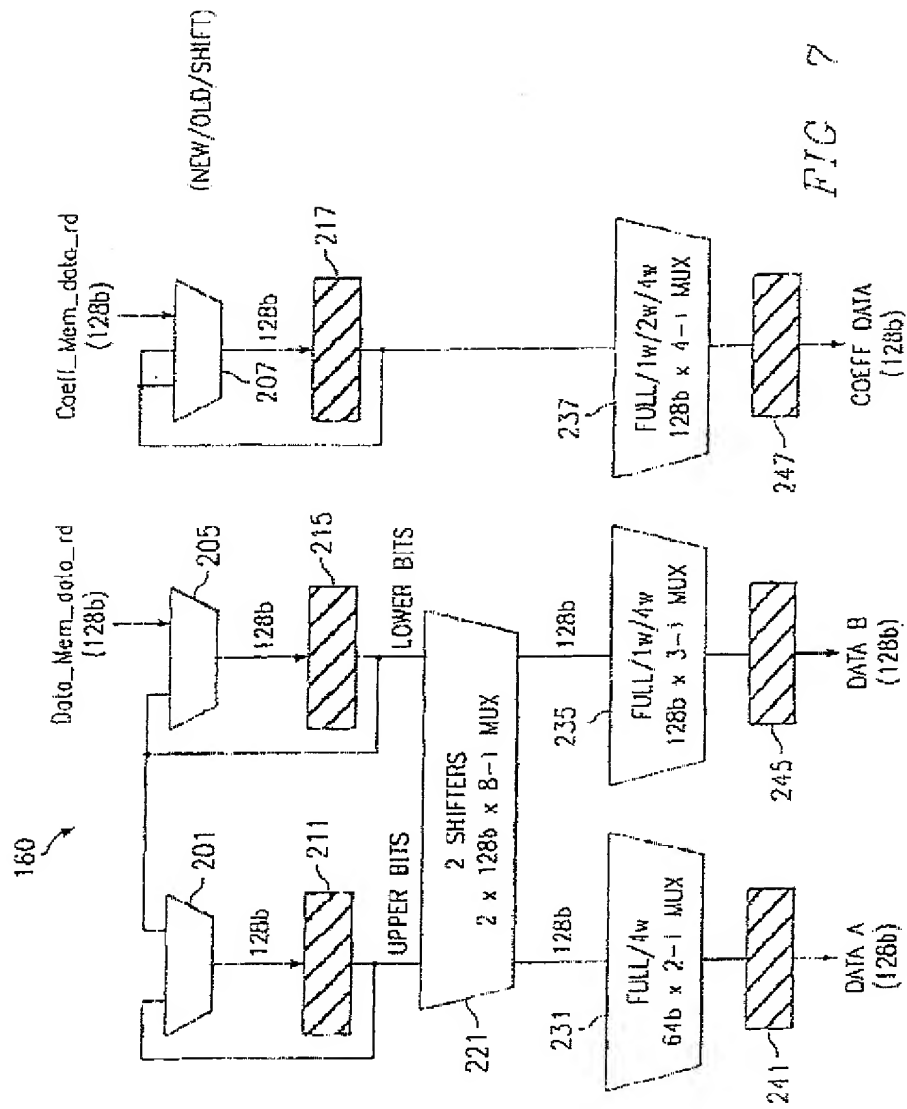
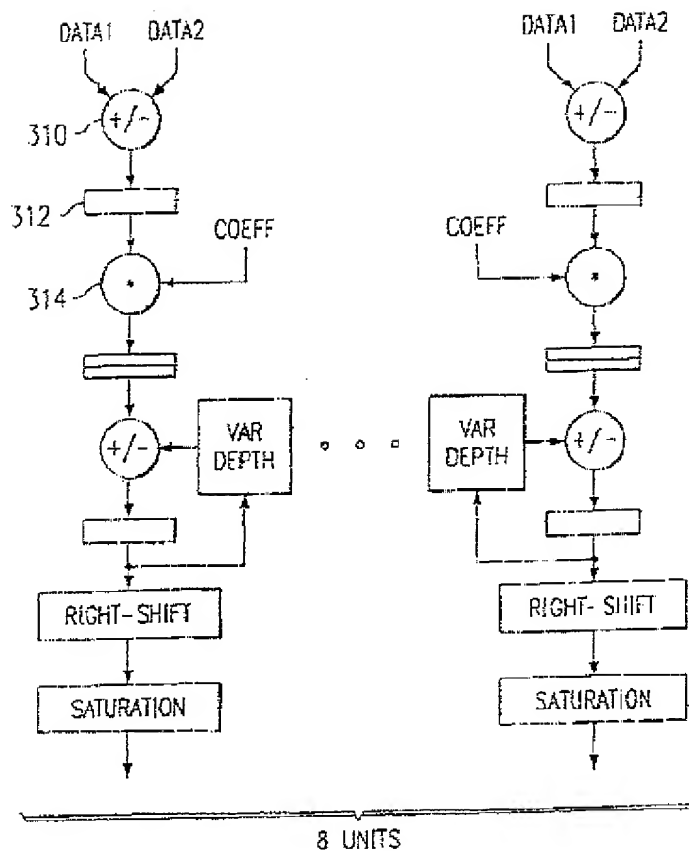
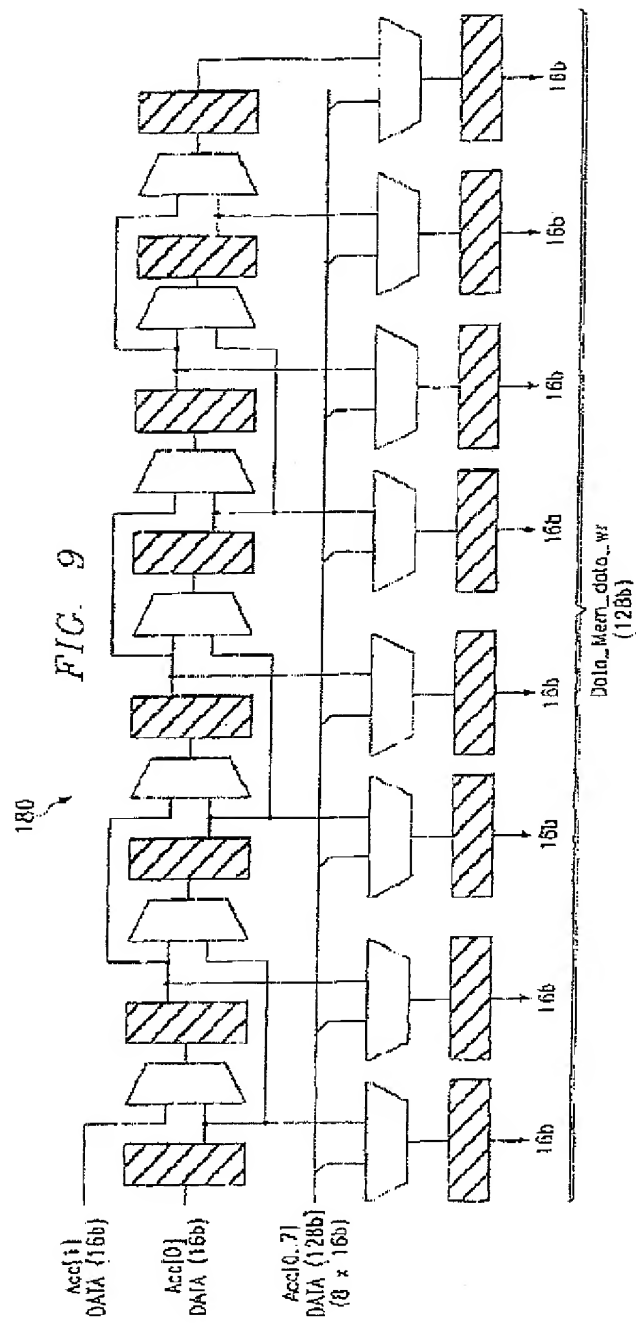


FIG 7

FIG. 8





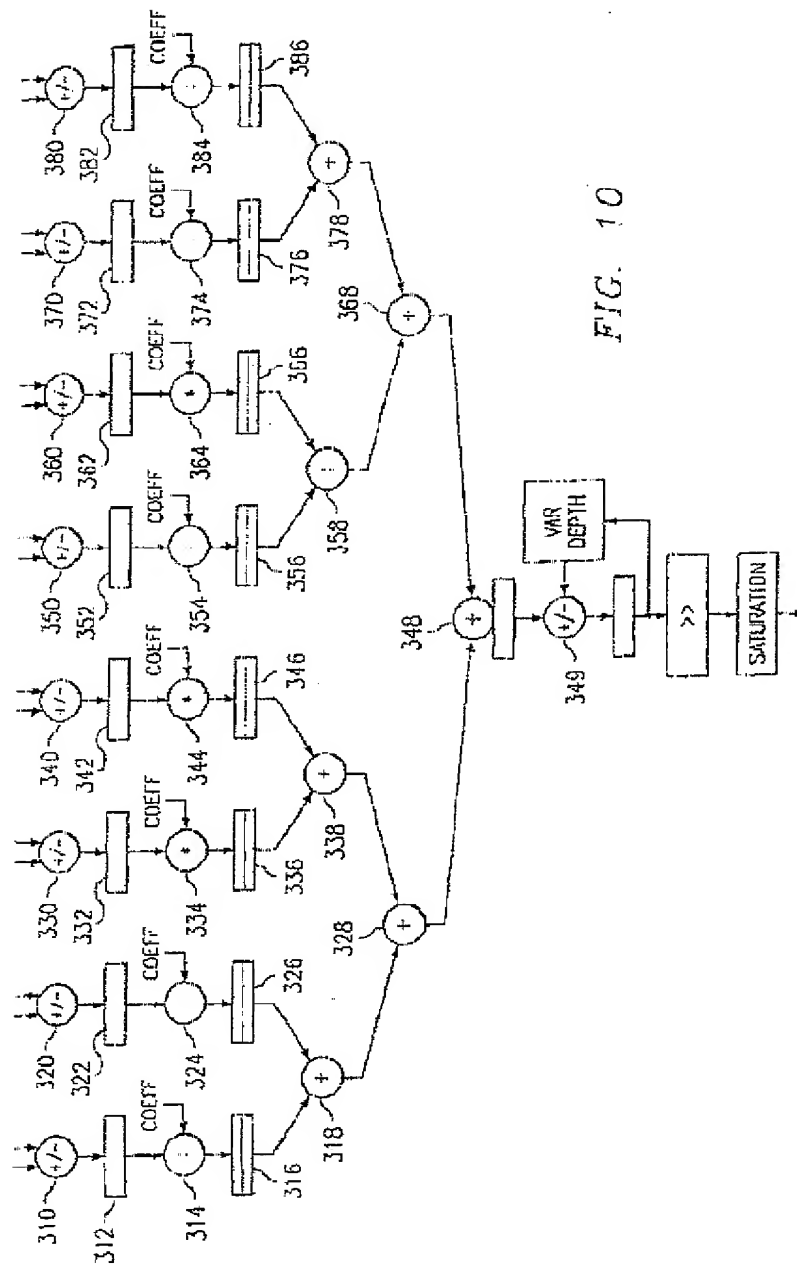


FIG. 10

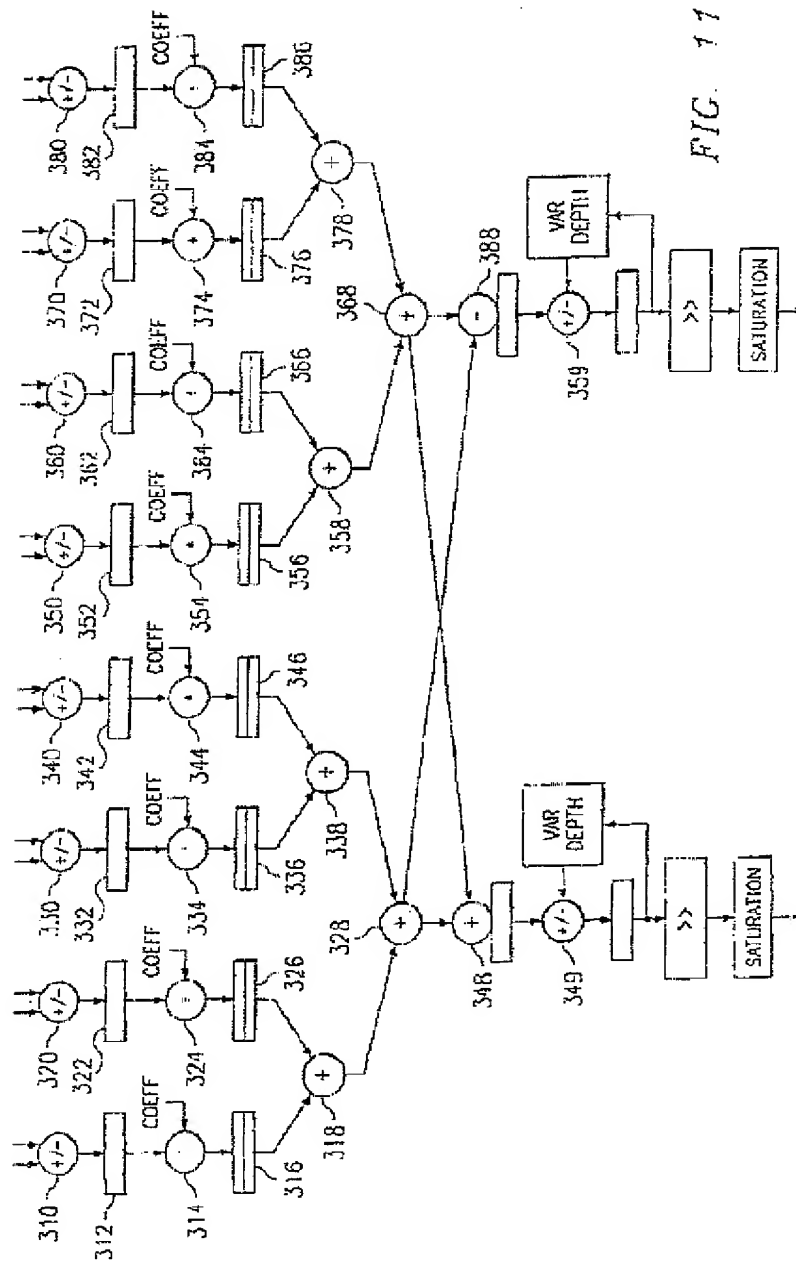


FIG. 11

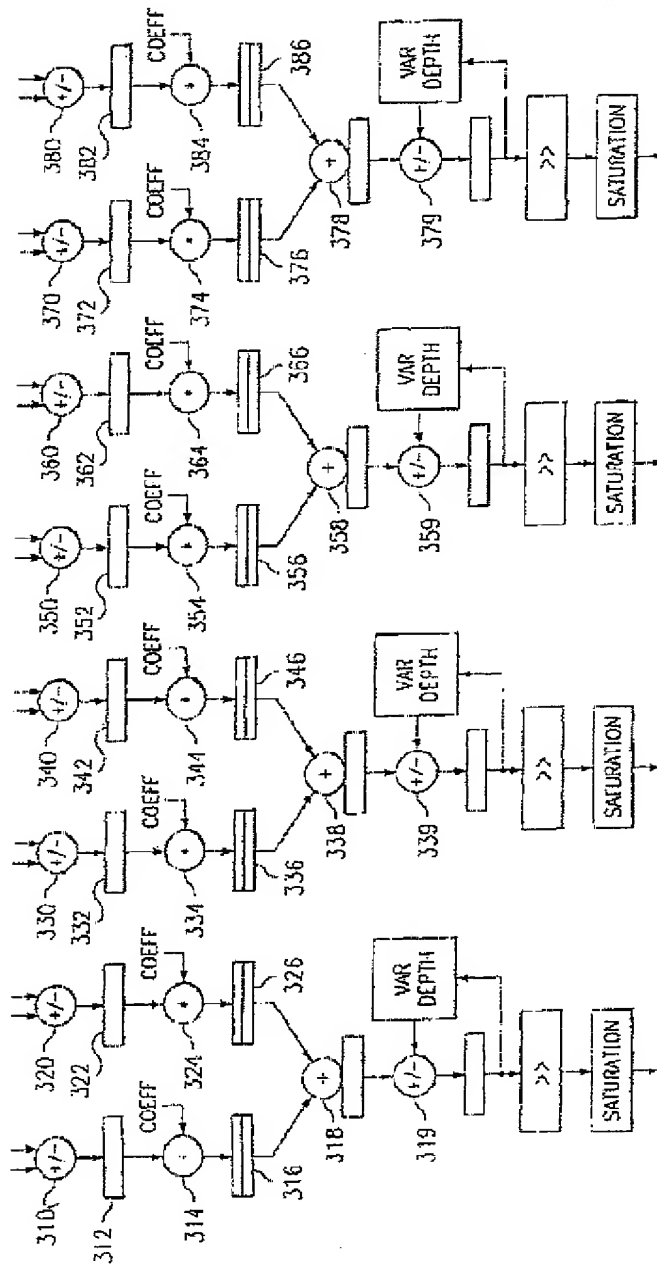


FIG. 12

FIG 13a

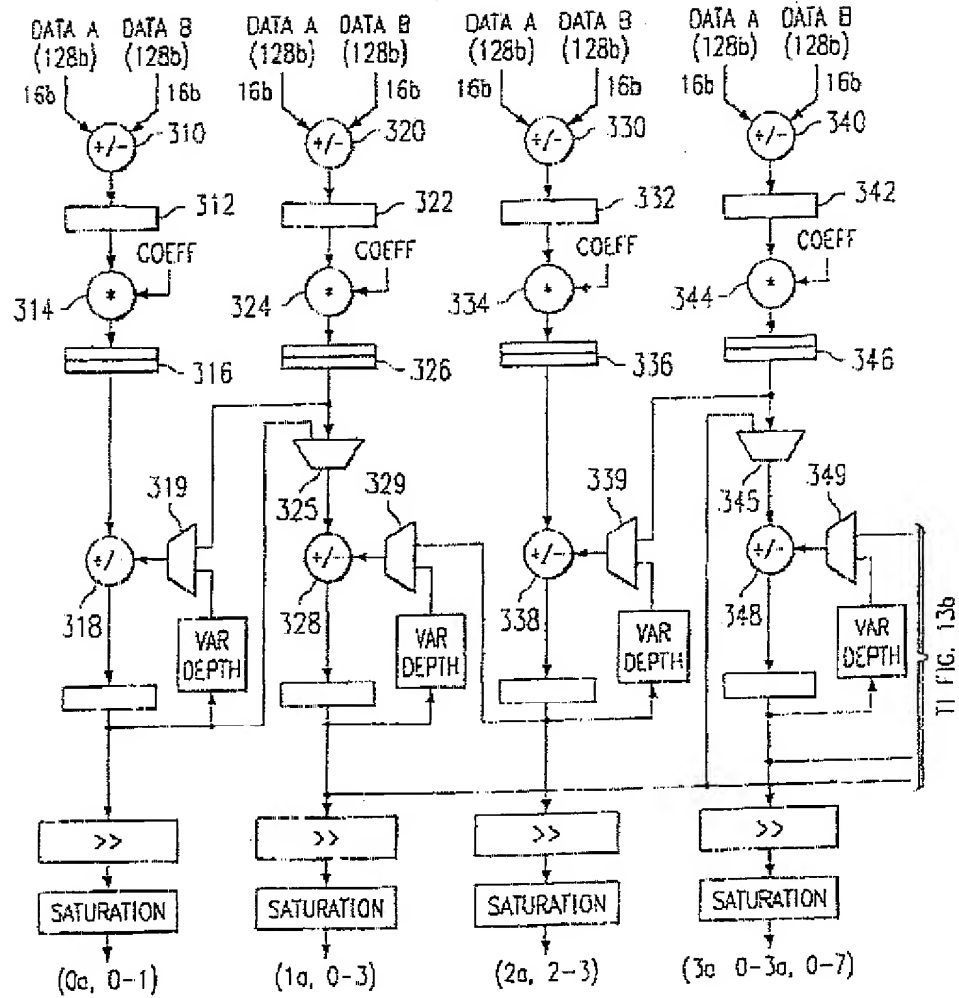
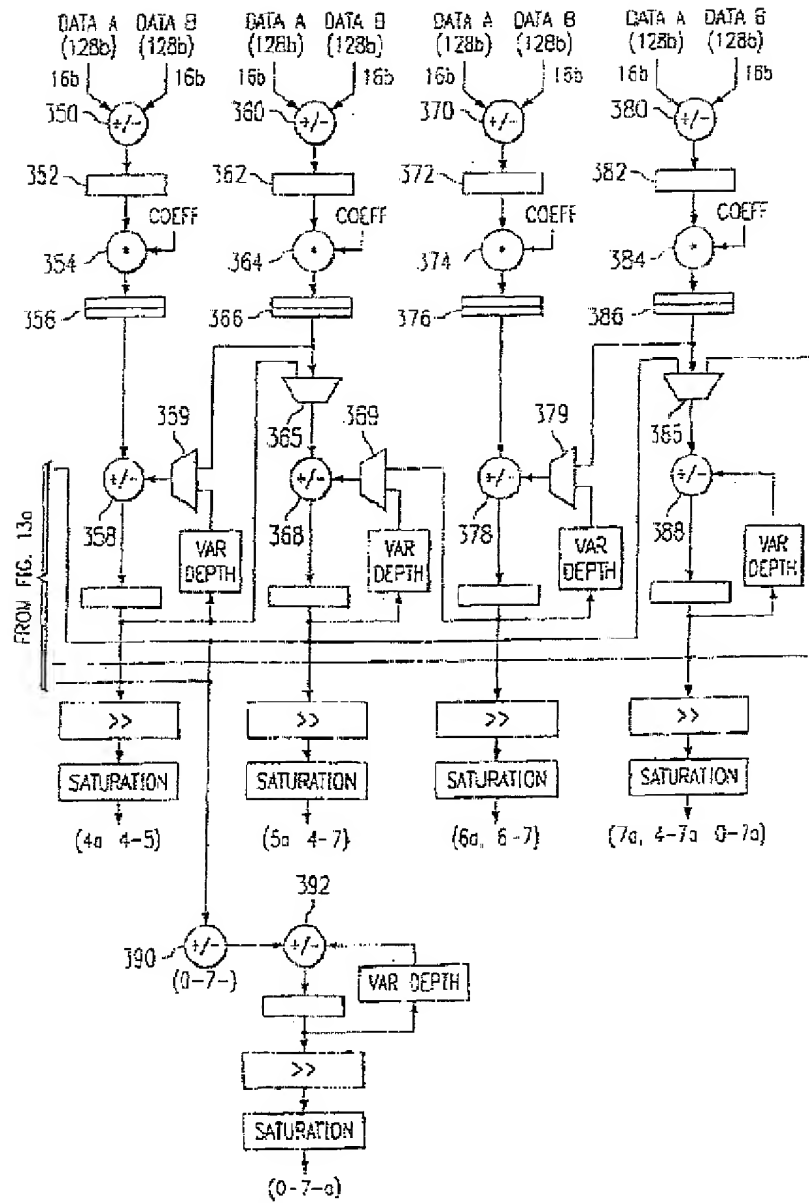




FIG. 13b



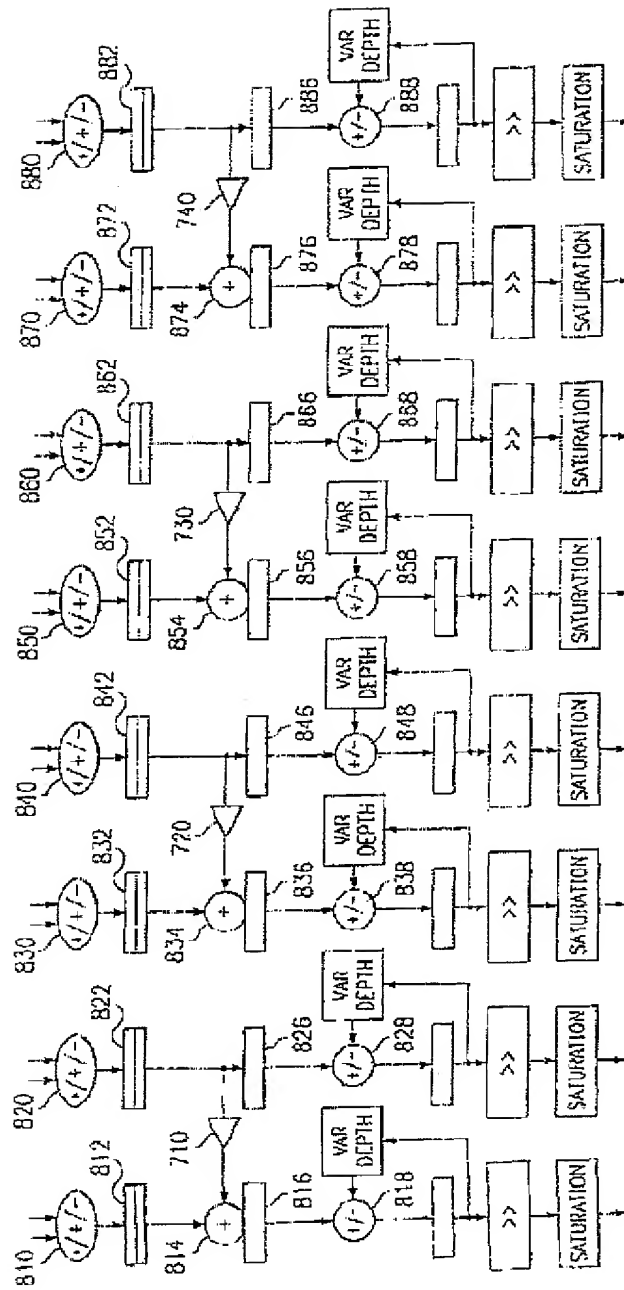


FIG 14

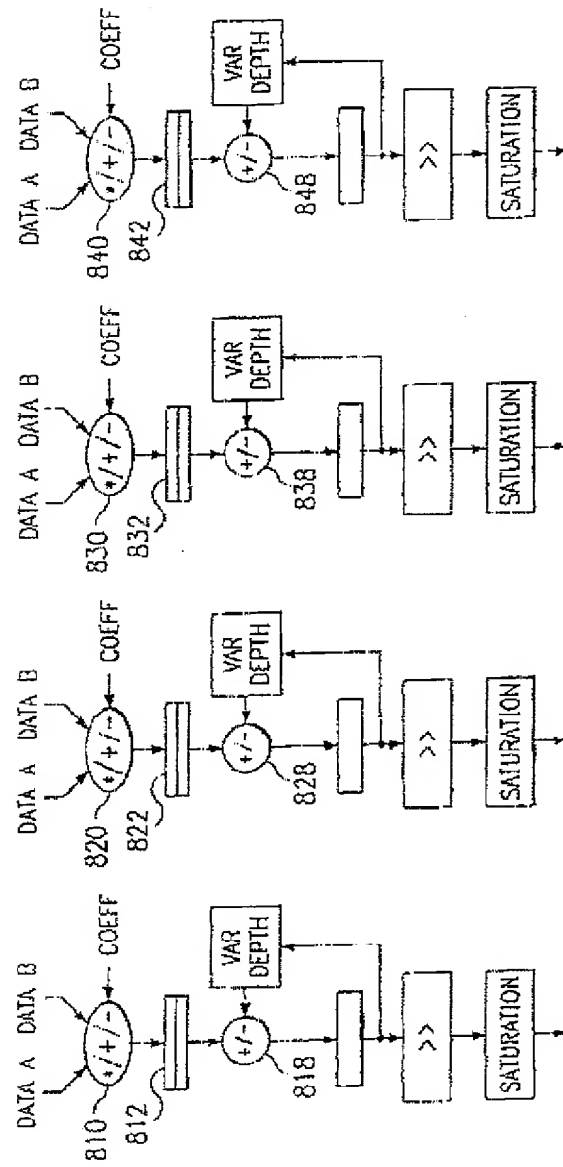


FIG. 15

FIG 16

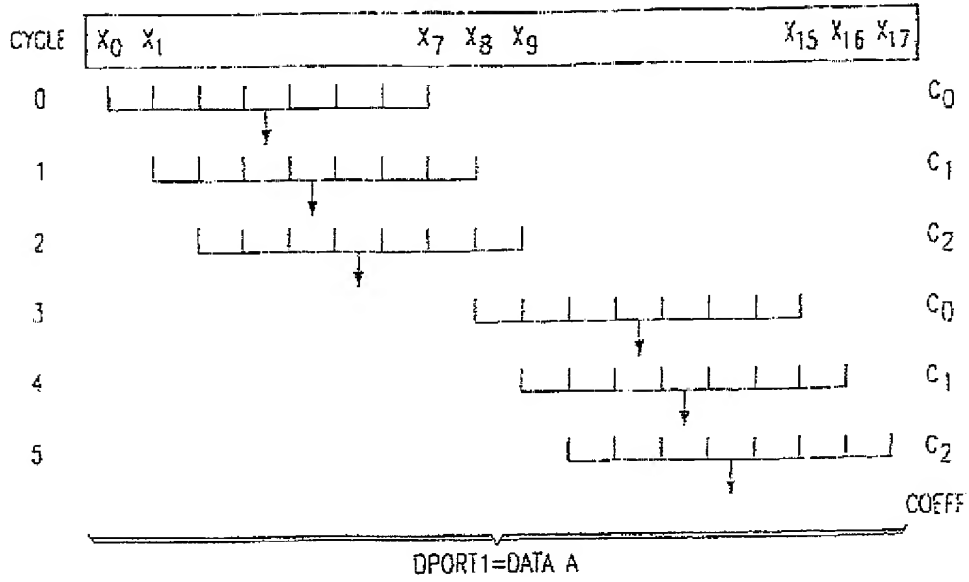


FIG 17

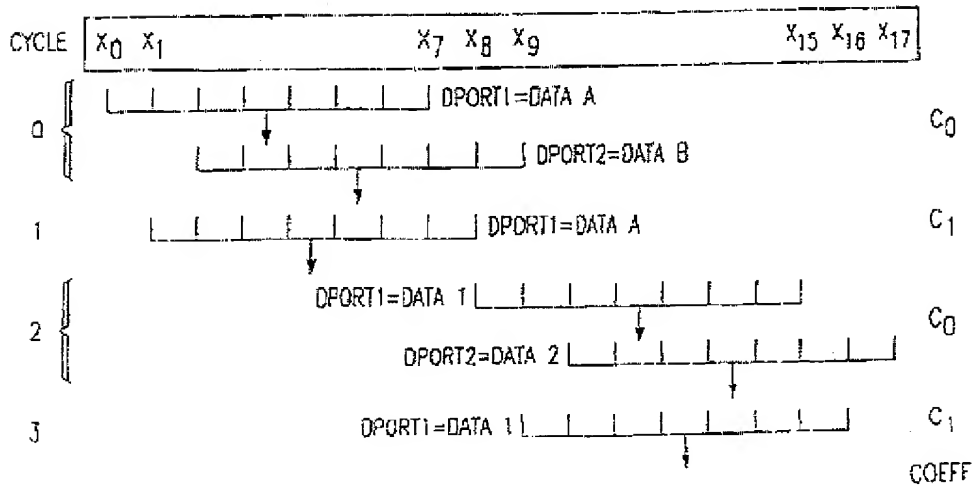
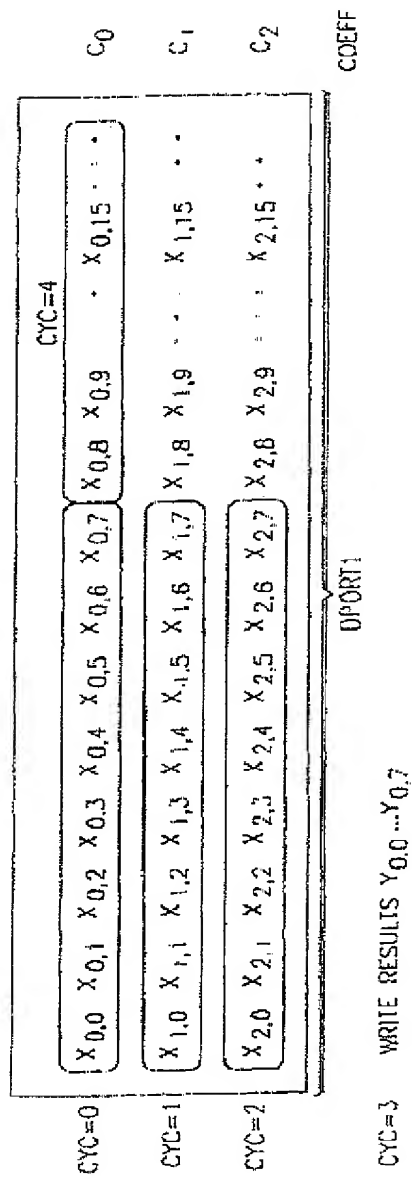


FIG. 18



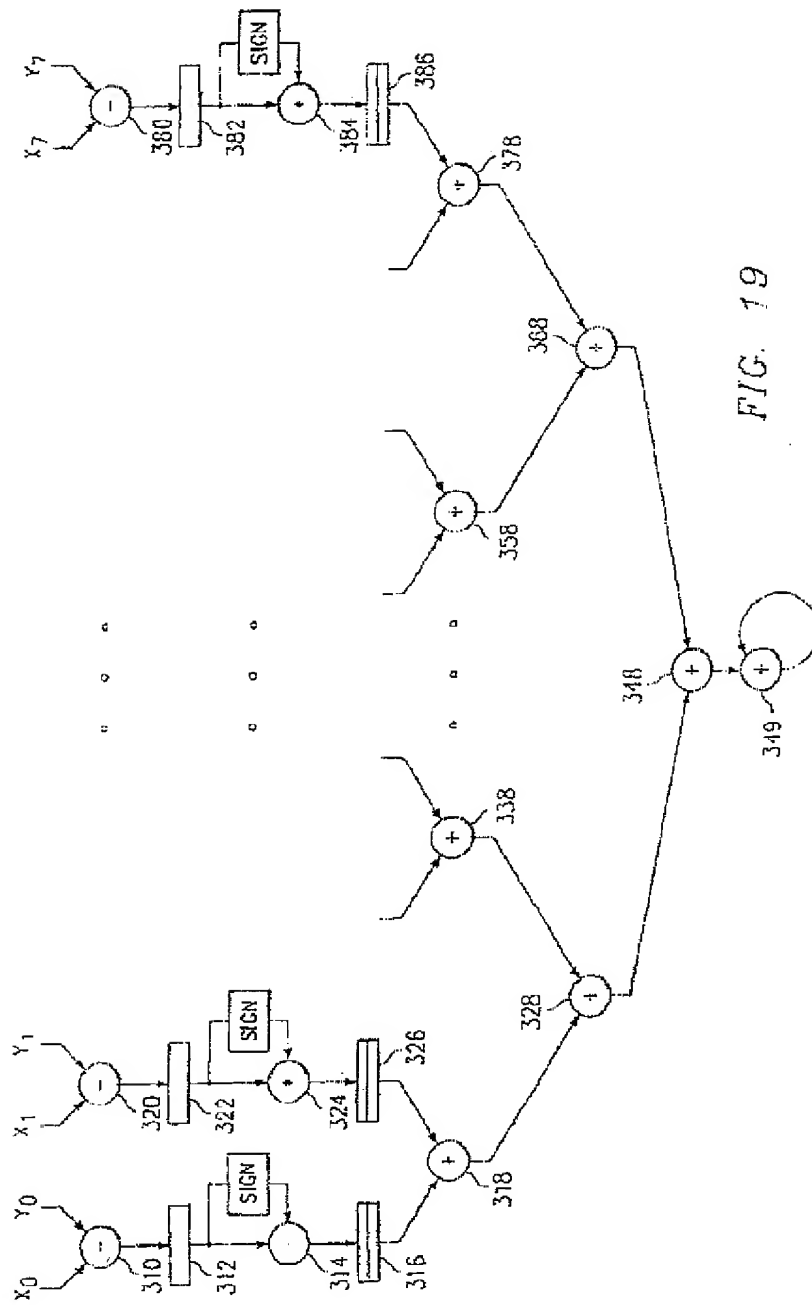


FIG. 20

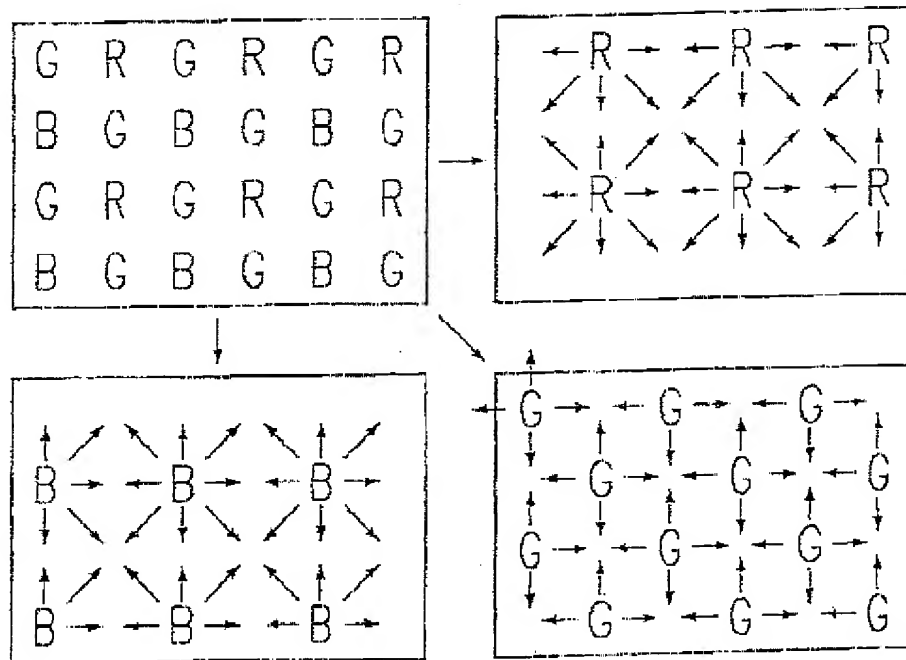


FIG. 21

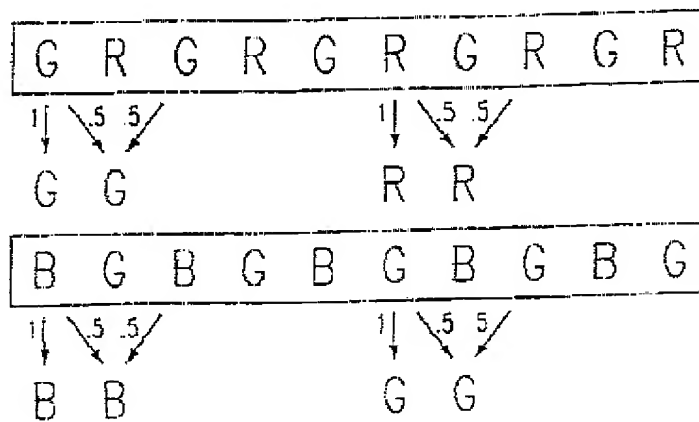


FIG. 22a

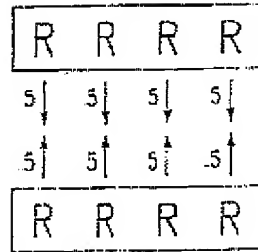


FIG. 22b

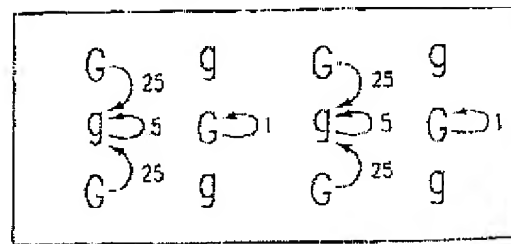


FIG. 23

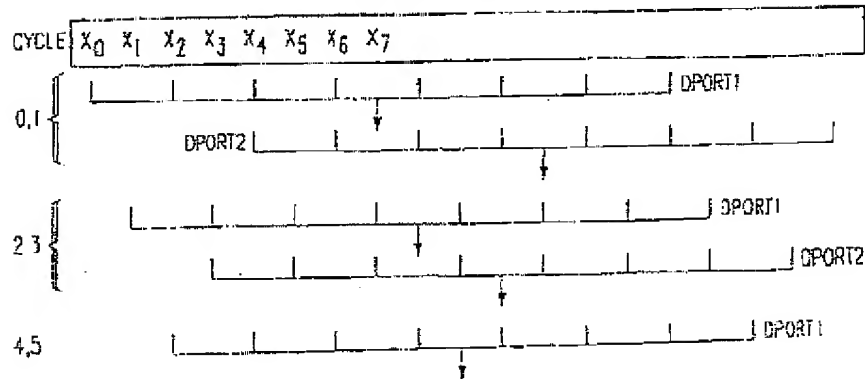




FIG. 24

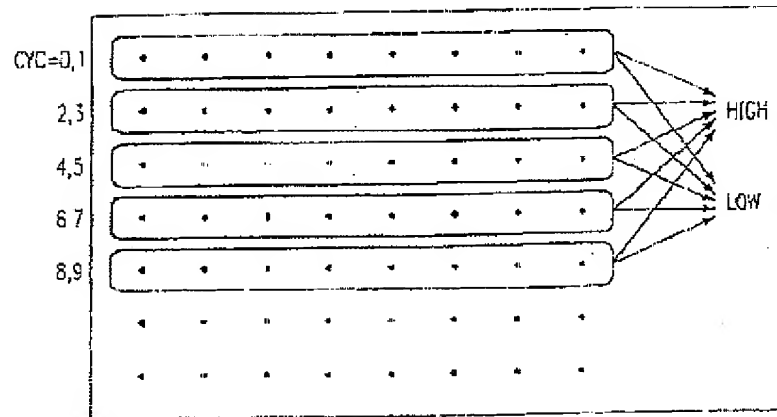


FIG. 25

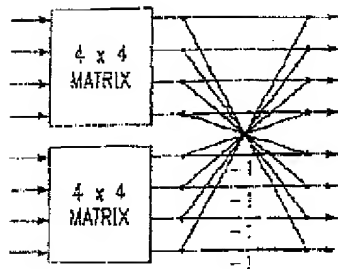


FIG. 26

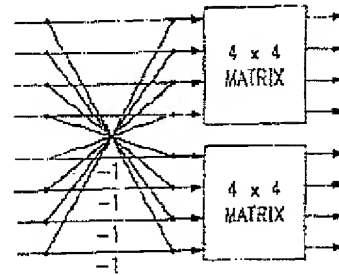


FIG. 27

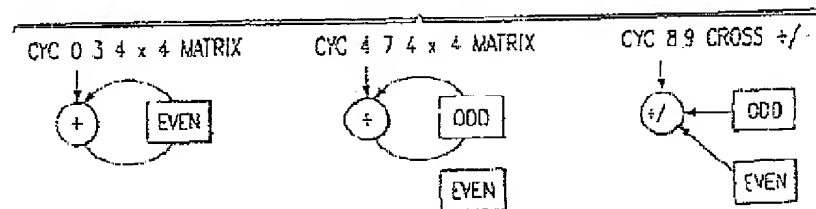
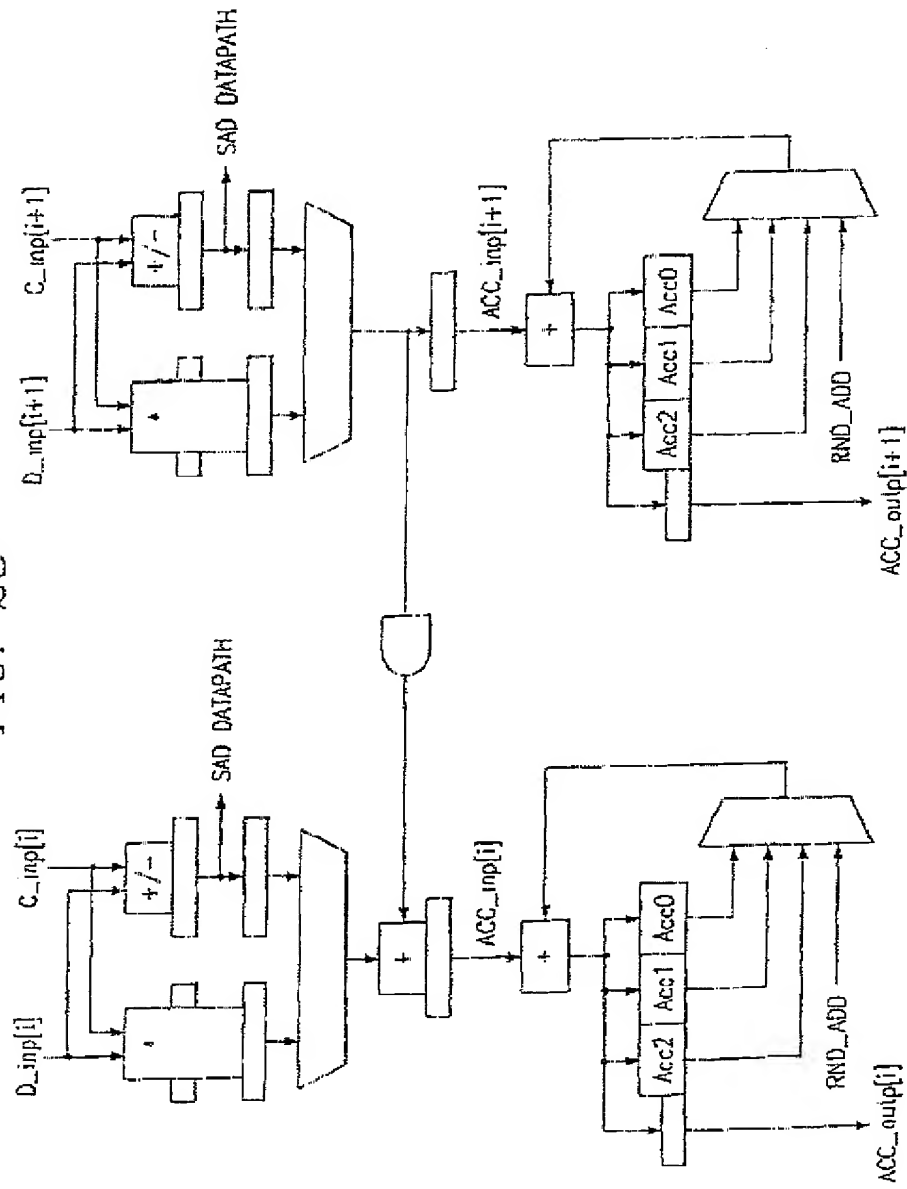


FIG. 28



5

**ABSTRACT**

The proposed architecture is integrated onto a Digital Signal Processor (DSP) as a coprocessor (140) to assist in the computation of sum of absolute differences, symmetrical row/column Finite Impulse Response (FIR) filtering with a downsampling (or upsampling) option, row/column Discrete Cosine Transform (DCT)/Inverse Discrete Cosine Transform (IDCT), and generic algebraic functions. The architecture is called IPP, which stands for image processing peripheral, and consists of 8 multiply-accumulate hardware units connected in parallel and routed and multiplexed together. The architecture can be dependent upon a Direct Memory Access (DMA) controller (120) to retrieve and write back data from/to DSP memory without intervention from the DSP core (110). The DSP can set up the DMA transfer and IPP/DMA synchronization in advance, then go on its own processing task. Alternatively, the DSP can perform the data transfers and synchronization itself by synchronizing with the IPP architecture on these transfers. This architecture implements 2-D filtering, symmetrical filtering, short filters, sum of absolute differences, and mosaic decoding more efficiently than the previously disclosed architectures of the prior art.

20